Digital Image Processing

SEGMENTATION

THRESHOLDING

- ***Segmentation*** is the process of partitioning a digital image into multiple segments (sets of pixels, also known as super pixels).
- The goal of segmentation is to simplify and/or change the representation of an image into something that is more meaningful and easier to analyze.
- Image segmentation is typically used to locate objects and boundaries (lines, curves, etc.) in images.
- More precisely, image segmentation is the process of assigning a label to every pixel in an image such that pixels with the same label share certain visual characteristics.
- **Complete segmentation** - set of disjoint regions uniquely corresponding with objects in the input image.
- Cooperation with higher processing levels which use specific knowledge of the problem domain is necessary.
- **Partial segmentation** - regions do not correspond directly with image objects.
- Image is divided into separate regions that are homogeneous with respect to a chosen property such as brightness, color, reflectivity, texture, etc.
- In a complex scene, a set of possibly overlapping homogeneous regions may result. The partially segmented image must then be subjected to further processing, and the final image segmentation may be found with the help of higher level information.
- Simple segmentation problems:
    - contrasted objects on a uniform background
    - Simple assembly tasks, blood cells, printed characters, etc.
- Totally correct and complete segmentation of complex scenes usually cannot be achieved in this processing phase.
- A reasonable aim is to use partial segmentation as an input to higher level processing.
- Segmentation problems:
    - image data ambiguity

    - information noise
- Segmentation methods
    - global approaches, e.g. using histogram of image features
    - edge-based segmentations
    - region-based segmentations
- Characteristics used in edge detection or region growing are brightness, texture, velocity field, etc...

Thresholding

- The simplest method of image segmentation is called the thresholding method. This method is based on a clip-level (or a threshold value) to turn a gray-scale image into a binary image.
- The key of this method is to select the threshold value (or values when multiple-levels are selected). Several popular methods are used in industry including the maximum entropy method, **Otsu's method** (maximum variance), and **k-means** clustering.

[Threshold–> Strip of wood or stone forming the bottom of a doorway and crossed in entering a house or room.
A level or point at which something starts or ceases to happen or come into effect. The level at which one starts to feel or react to something.]

- The purpose of thresholding is to extract those pixels from some image which represent an object (either text or other line image data such as graphs, maps).
- Though the information is binary the pixels represent a range of intensities.
- Thus the objective of binarization is to mark pixels that belong to true foreground regions with a single intensity and background regions with different intensities.
- Thresholding is the transformation of an input image *f* to an output (segmented).

  Binary image *g* as follows

  g(i,j)=1 for *f(i,j)* >=T.

g(i,j)=0 for *f(i,j)* <T.

- Where T is the thresholding, *g(i,j)*=1 for image elements of objects, and *g(i,j)*=0 for image elements of background
- If objects do not touch each other, and if their gray-levels are clearly distinct from background gray-levels, thresholding is suitable segmentation method.
- Correct threshold selection is crucial for successful threshold segmentation; this selection can be determined interactively or it can be the result of some threshold detection method.
- Only under very unusual circumstances can threshold be successful using a single threshold for whole image(global thresholding) since even in very simple images there are likely to be gray-level variations in objects and background; this variation may be due to non-uniform lighting, non-uniform input device parameters or a number of other factors.
- Segmentation using variable thresholds (adaptive thresholding), in which the threshold value varies over the image as a function of local image characteristics.

- A global threshold is determined from the whole image *f :*

$$T=T(f)\ .$$

- Local Thresholds are position dependent

$$T=T(f,f_c),$$

- Where f_c is that image part in which the threshold is determined. One option is to divide the image *f into* sub images f_c and determine a threshold independently in each subimages, it can be interpolated from thresholds determined in neighboring sub images. Each sub images is then processed with respect to its local threshold.

- Basic thresholding as defined has many modifications. One possibility is to segment an image into regions of pixels with gray-levels from a set D and into background otherwise (band thresholding):

  $g(i,j)=1$ for $f(i,j) \in D$

  $g(i,j)=0$ otherwise.

- This thresholding can be useful, for instance, in microscopic blood cell segmentations, where a particular gray-level interval represents Cytoplasma, the background is lighter, and the cell kernel darker.
- This thresholding definition can serve as a border detector as well; assuming dark objects on a light background, some gray-levels between those of objects and background can be found only in the object borders.

Thresholding algorithms

- For a thresholding algorithm to be really effective, it should preserve logical and semantic content. There are two types of thresholding algorithms
  - Global thresholding algorithms
  - Local or adaptive thresholding algorithms
- In global thresholding, a single threshold for all the image pixels is used.
- When the pixel values of the components and that of background are fairly consistent in their respective values over the entire image, global thresholding could be used.
- In adaptive thresholding, different threshold values for different local areas are used.

Basic Global Thresholding

- Simplest of all thresholding techniques is to partition the image histogram by using a single global threshold, T.
- Segmentation is then accomplished by scanning the image pixel by pixel and labeling each pixel as object or background, depending on whether the gray level of that pixel is greater or less then the value of T.

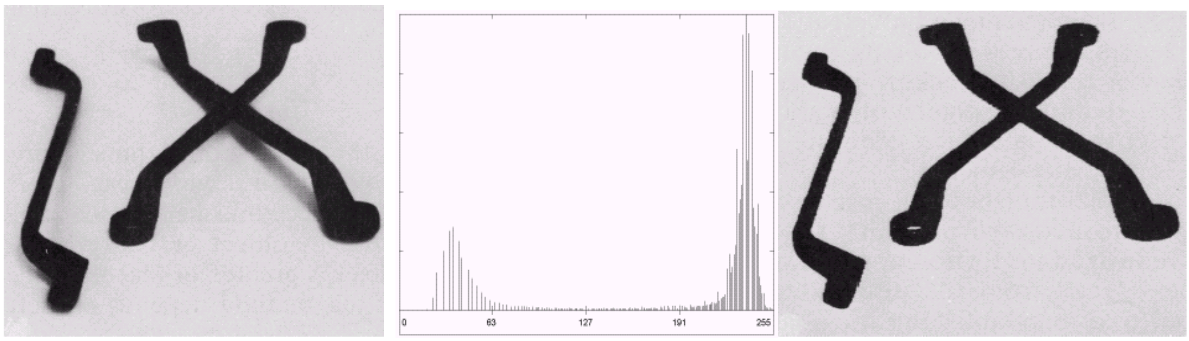


Fig 1(a)Original image (b)Image histogram (c)Result of global thresholding with T midway between the maximum and minimum gray levels

- Fig 1(a) shows a simple image and Fig 1(b) shows its histogram, Fig1(c) shows the result of segmentation Fig 1 by using a threshold T midway between the maximum and minimum gray levels.
- This Threshold achieved a clean segmentation by eliminating the shadows and leaving only the objects themselves. The objects of interest in this case are darker than the background, so any pixel with a gray level<=T was labeled black (0) and any pixel with gray level>T was labeled white (255).
- The key objective is merely to generate a binary image, so the black-white relationship could be reversed.

Algorithm: Basic Global Thresholding

The following algorithm has been used to obtain the initial threshold T, in the following section automatically:

1. Choose initial estimate for T.
2. Divide the histogram using T. This will produce 2 groups of pixels:
   - G_1 - Set of all pixels with grey level values >T.
   - G_2 - Set of pixels with values <=T.
3. Compute the average grey level values m_1 and m_2 for the pixels in regions G_1 and G_2.
4. Compute a new threshold value:

$$T=1/2(m_1+m_2)$$

Repeat steps 2 through 4 until the difference in T in successive iterations is smaller than a predefined parameter T_0.

Basic Adaptive Thresholding:

- Single value thresholding will not work to divide an image into sub images and threshold these individually.
- An approach for handling such a situation is to divide the original image into sub images and then utilize a different threshold to segment each sub image.
- The key issues in this approach are how to subdivide the image and how to estimate the threshold for each resulting sub image. Since the threshold used for each pixel depends on the location of the pixel in terms of the sub images, this type of thresholding is adaptive.

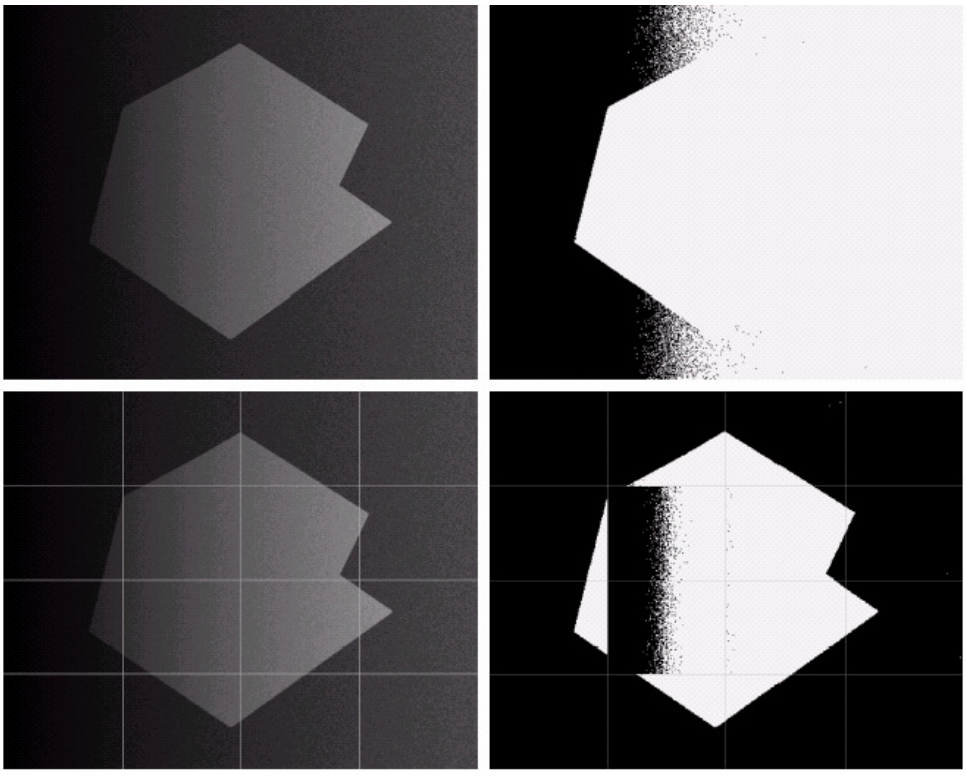


Fig 2: (a) Original Image (b) Result of global thresholding (c)Image subdivided into individual sub images (d)Result of adaptive thresholding

- Fig 2(a) which we concluded could not be thresholded effectively with a single global threshold previously.
- Fig 2(b) shows the result of thresholding the image with a global threshold manually placed in the valley of its histogram.

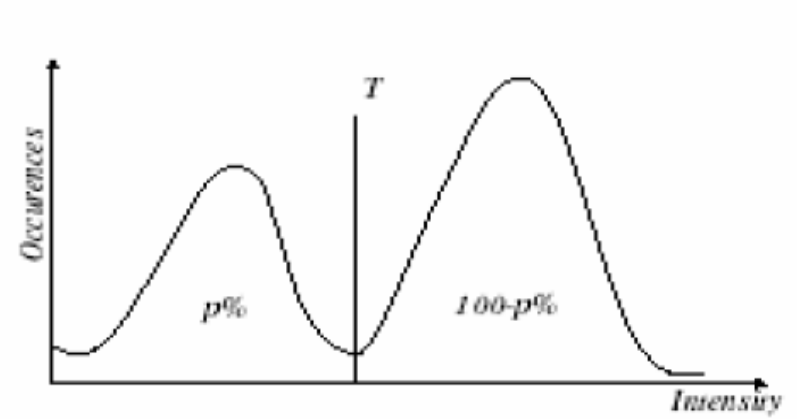
- One approach to reduce the effect of non uniform illumination is to subdivide the image into smaller sub images, such that the illumination of each sub image is approximately uniform.
- Fig 2(c) shows such a partition, obtained by subdividing the image into four equal parts, and then subdividing each part by four again.
- All the sub images that did not contain a boundary between object and background had variances of less than 75; All sub images containing boundaries had variances in excess of 100.
- Each sub image with variance greater than 100 was segmented with a threshold computed for that sub image using the algorithm.
- The initial value for T in each case was selected as the point midway between the minimum and maximum gray levels in the sub image.
- All sub images with variance less than 100 were treated as one composite image, which was segmented using a single threshold estimated using the same algorithm.
- The result of segmentation using this procedure is shown in Fig 2(d).

Threshold Detection Methods

P-Tile (Percentile) Thresholding:

- This Technique uses knowledge about the area size of the desired object to the threshold an image.
- If some property of an image after segmentation is known a priori, the task of threshold selection is simplified, since the threshold is chosen to ensure that this property is satisfied.
- A printed text sheet may be an example if we know that character of the text cover 1/p of the sheet area. Using this prior information about the ratio between the sheet area and character area, it is very easy to choose a threshold T (based on the image histogram) such that 1/p of the image area has gray values less than T and the rest has gray values larger than T. This method is called P-tile thresholding.

- P-tile Method:- If object occupies P% of image pixels the set a threshold T such that P% of pixels have intensity below T.

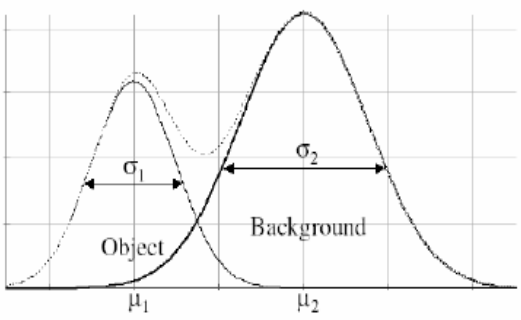


Histogram Shape Analysis:-

- If the image consists of objects of approximately the same gray-level that differs from the gray-level of the background, the resulting histogram is **bi-modal**.
- Pixels of objects form one of its peaks, while pixels of the background form the second peak.
- The histogram shape illustrates the fact that the gray values between the two peaks are not common in the image, and probably result from border pixels between objects and background.
- The chosen threshold must meet minimum segmentation error requirements; it makes intuitive sense to determine the threshold as the gray-level that has a minimum histogram value between the two mentioned maxima.
- If the histogram is multi-modal, more thresholds may be determined at minima between any two maxima. Each threshold gives different segmentation results.

Mode Method:-

- To decide if the histogram is bi-modal or multi-modal may not be so simple in reality, it often being impossible to interpret the significance of local histogram maxima.
- Bi-modal histogram threshold detection algorithms usually find the highest local maxima first and detect the threshold as a minimum between them; this technique is called the mode method.
- To avoid detection of two local maxima belonging to the same global maximum, a minimum distance in gray-levels between these maxima is usually required, or techniques to smooth histograms are applied.
- Assume that gray values are drawn from two normal distributions with parameters $(\mu_1, \sigma_1), (\mu_2, \sigma_2)$
- If the standard deviations are zero, there will be two spikes in the histogram and the threshold can be placed anywhere between them.
- For non-ideal cases, there will be peaks and valleys and the threshold can be placed corresponding to the valley



Optimal Thresholding:-

- Methods based on approximation of the histogram of an image using a weighted sum of two or more probability densities with normal distribution represent a different approach called **optimal thresholding**.

- The threshold is set as the closest gray-level corresponding to the minimum probability between the maxima of two or more normal distributions, which results in minimum error segmentation (the smallest number of pixels is mis-sementated).
- See **Fig 3** the difficulty with these methods is in estimating normal distribution parameters together with the uncertainty is sought that maximizes gray-level variance between objects and background.
- Note that this approach can be applied even if more than one threshold is needed.
- Alternatively, minimization of variance of the histogram, sum of square errors, spatial entropy, average clustering, or other optimization approaches may be used.

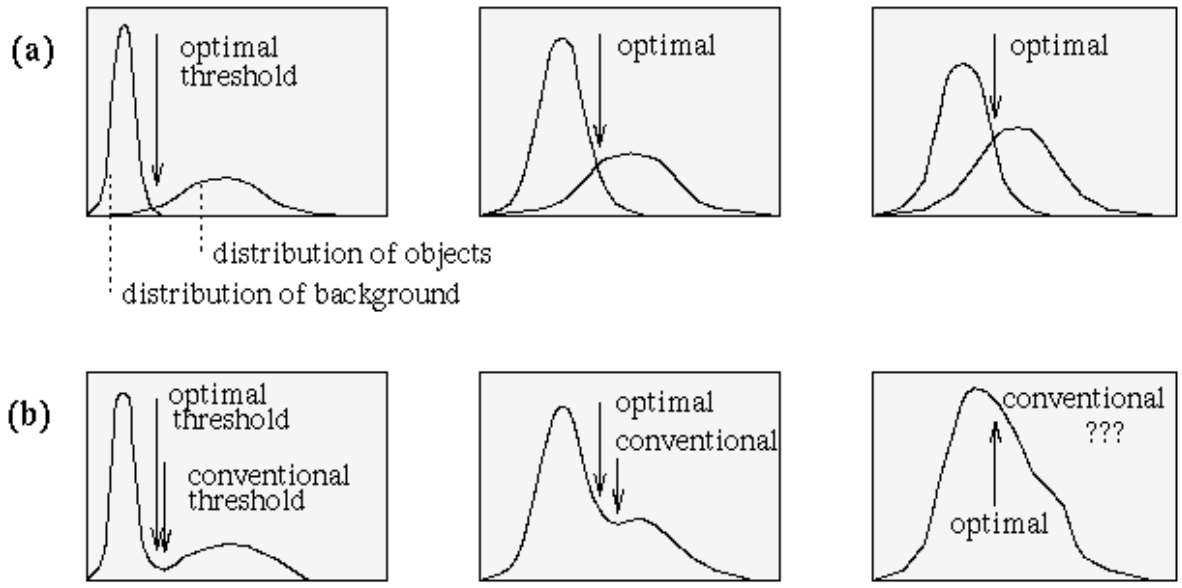


Fig 3: Gray-level histograms approximated by two normal distributions—the threshold is set to give minimum probability of segmentation error. (a)Probability distributions of background and objects . (b)Corresponding histograms and optimal threshold

- The following algorithm represents a simpler version that shows a rationale for this approach and works well even if the image histogram is not bi-modal.

- This method assumes that regions of two main gray-levels are present in the image, thresholding of printed text being an example. The algorithm is iterative, four to ten iterations usually being sufficient.

Algorithm Iterative (optimal) Threshold selection:-

1. Assume no knowledge about the exact location of objects, consider as a first approximation that the four corners of the image contain background pixels only and the remainder contains object pixels.
2. At step t, compute μ_B^t and μ_0^t as the mean background and object gray level, respectively, where segmentation into background and objects at step t is defined by the threshold value T^t determined in the previous step

$$\mu_B^t = \frac{\sum_{(i,j)\in background} f(i,j)}{\#background\_pixels} \qquad \mu_0^t = \frac{\sum_{(i,j)\in objects} f(i,j)}{\#objects\_pixels}$$

3. Set

$$T^{(t+1)} = \frac{\mu_B^t + \mu_0^t}{2}$$

$T^{(t+1)}$ now provides an updated background---object distinction.

4. If $T^{(t+1)} = T^t$, halt; otherwise return to step 2

Segmentation: Edge-based segmentation

Edge-based Segmentation

- An edge is the boundary between an object and the background.
- Edge-based segmentation represents a large group of methods based on information about edges in the image
- Edge-based segmentations rely on edges found in an image by edge detecting operators -- these edges mark image locations of discontinuities in gray level, color, texture, etc.
- Image resulting from edge detection cannot be used as a segmentation result.
- Supplementary processing steps must follow to combine edges into edge chains that correspond better with borders in the image.
- The final aim is to reach at least a partial segmentation -- that is, to group local edges into an image where only edge chains with a correspondence to existing objects or image parts are present.
- The more prior information that is available to the segmentation process, the better the segmentation results that can be obtained.
- The most common problems of edge-based segmentation are
    - an edge presence in locations where there is no border, and
    - no edge presence where a real border exists.

Edge image thresholding

- Almost no zero-value pixels are present in an edge image, but small edge values correspond to non-significant gray level changes resulting from quantization noise, small lighting irregularities, etc. Simple Thresholding of an edge image can be applied to remove these small values. The approach is based on an image of edge magnitudes processed by an appropriate threshold.
- Selection of an appropriate global threshold is often difficult and sometimes impossible; p-tile thresholding can be applied to define a threshold
- Alternatively, non-maximal suppression and hysteresis thresholding can be used as was introduced in the Canny edge detector.

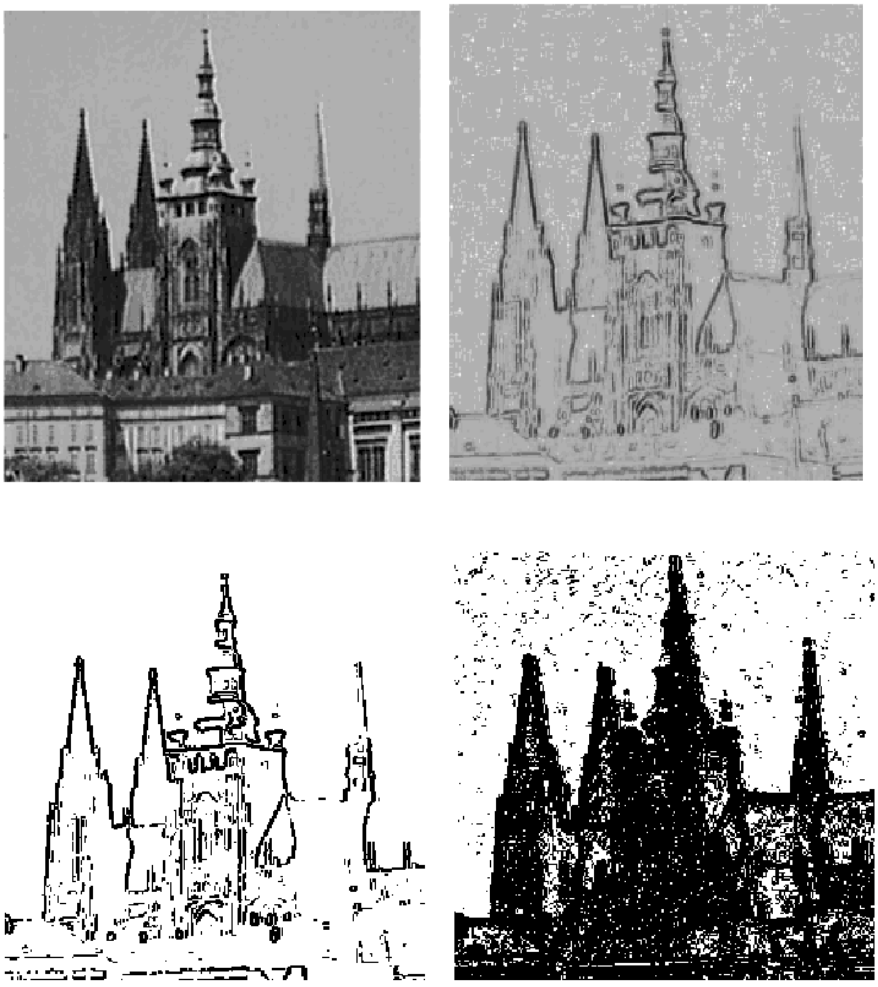


Fig 1: Edge image thresholding (a)Original image (b)Edge image (c)Edge image thresholding at 30 (d)Edge image thresholding at 10

Algorithm : Non Maximal suppression of directional edge data

1. Quantize edge directions eight ways according to 8-connectivity

2. For each pixel with non-zero edge magnitude, inspect the two adjacent pixels indicated by the direction of its edge

3. If the edge magnitude of either of these two exceeds that of the pixel under inspection, mark it for deletion.

4. When all pixels have been inspected, re-scan the image and erase to zero all edge data marked for deletion.

This algorithm is based on 8-connectivity and may be simplified for 4-coonectivity; it is also open to more sophisticated measurement of edge direction.

Algorithm: Hysteresis output of an edge detector

1. Mark all edges with magnitude greater than t_1 as correct.

2. Scan all pixels with edge magnitude in the range [t_0, t_1].

3. If such a pixel borders another already marked as an edge, then mark it too. 'Bordering' may be defined by 4- or 8- connectivity.

4. Repeat from step 2 until stability.

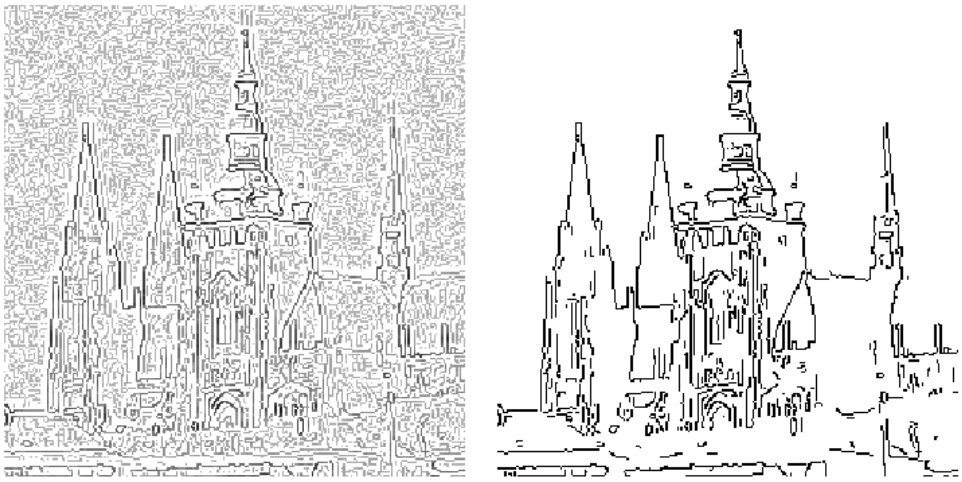


Fig 2 (a)Non-maximal suppression of the data (b)Hysteresis applied to (a).

Edge relaxation

- Borders resulting from the previous method are strongly affected by image noise, often with important parts missing.

- Considering edge properties in the context of their mutual neighbors can increase the quality of the resulting image.

- All the image properties, including those of further edge existence, are iteratively evaluated with more precision until the edge context is totally clear - based on the strength of edges in a specified local neighborhood, the confidence of each edge is either increased or decreased.

- A weak edge positioned between two strong edges provides an example of context; it is highly probable that this inter-positioned weak edge should be a part of a resulting boundary.
- If, on the other hand, an edge (even a strong one) is positioned by itself with no supporting context, it is probably not a part of any border.

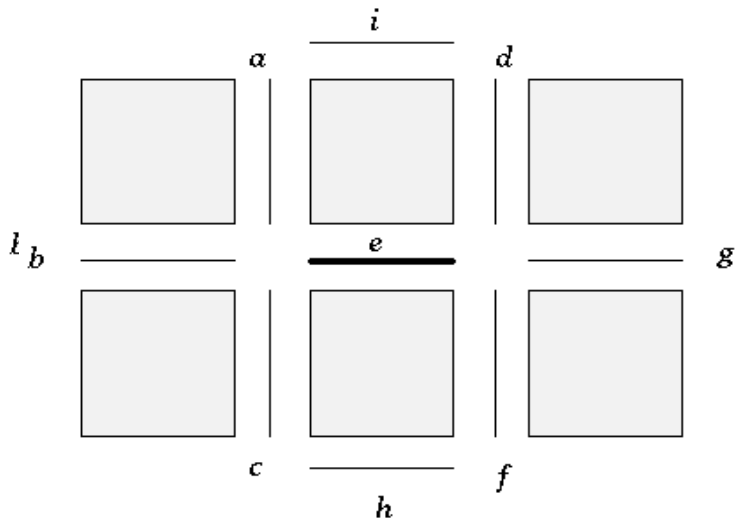


Fig 3 Crack edges surrounding central edge *e*

- Edge context is considered at both ends of an edge, giving the minimal edge neighborhood.

- The central edge e has a vertex at each of its ends and three possible border continuations can be found from both of these vertices.

- Vertex type -- number of edges emanating from the vertex, not counting the edge e.

- The type of edge e can then be represented using a number pair i-j describing edge patterns at each vertex, where i and j are the vertex types of the edge e.

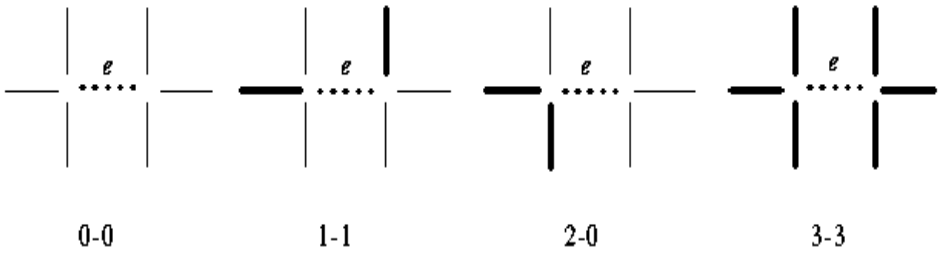


Fig 4 Edge patterns and corresponding edge types

0-0 isolated edge- negative influence on the edge confidence,
0-1 uncertain- weak positive, or no influence on edge confidence,
0-2,0-3 dead end- negati9ve influence on edge confidence,
1-1 continuation- strong positive influence on edge confidence,

1-2,1-3 continuation to border intersection- medium positive influence on edge confidence,
2-2,2-3,3-3 bridge between borders- not necessary for segmentation, no influence on edge confidence.

- Edge relaxation is an iterative method, with edge confidences converging either to zero (edge termination) or one (the edge forms a border).
- The confidence of each edge e in the first iteration can be defined as a normalized magnitude of the crack edge,
  - with normalization based on either the global maximum of crack edges in the whole image, or on a local maximum in some large neighborhood of the edge

Algorithm: Edge Relaxation

1.Evaluate a confidence $c^{(1)}$ (*e*) for all crack edges *e* in the image.

2. Find the edge type of each edge based on edge confidences $c^{(k)}$ (*e*) in its neighborhood.

3. Update the confidence $c^{(k+1)}$(*e*) of each edge *e* according to its type and its previous confidence $c^{(k)}$ (*e*).

4. Stop if all edge confidences have converged either to 0 or 1. Repeat steps 2 and 3 otherwise.

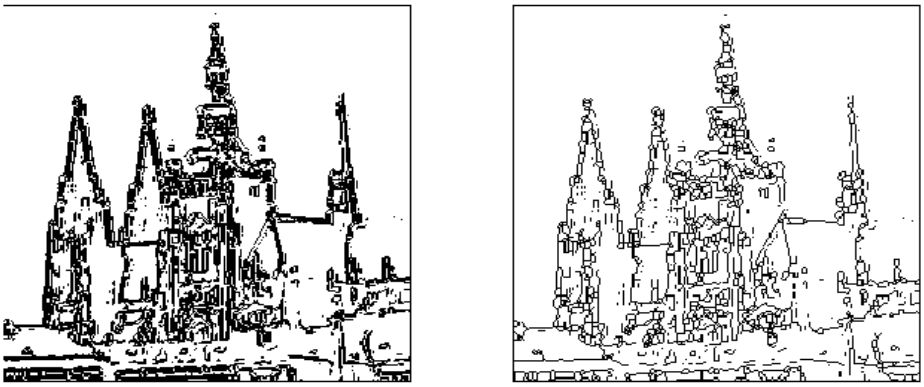
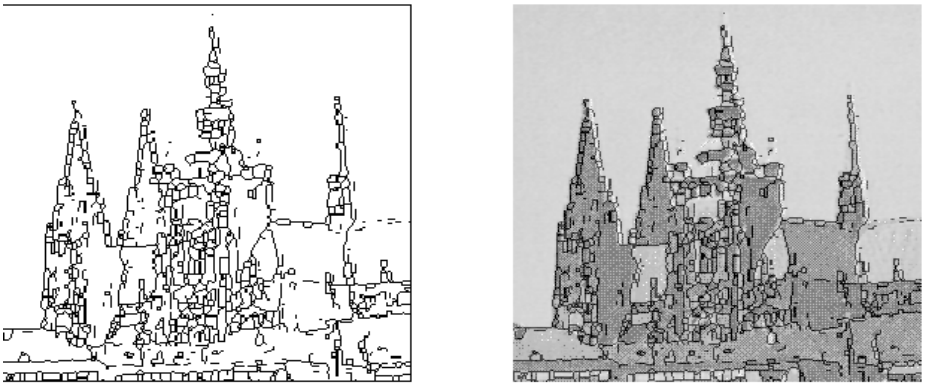


Fig 5 Edge relaxation (a) Resulting borders after 10 iterations (b) Borders after thinning.



(c) (d)

Fig 5 (c) Borders after 100 iterations, thinned (d) Borders after 100 iterations overlaid over original.

- The main steps of the above Algorithm are evaluation of vertex types followed by evaluation of edge types, and the manner in which the edge confidences are modified.

- A vertex is considered to be of type *i* if

$$type(i) = max_k\{type(k)\}, \qquad k = 0,1,2,3$$

$$type(0) = (m-a)(m-b)(m-c)$$

$$type(1) = a(m-b)(m-c)$$

$$type(2) = ab(m-c)$$

$$type(3) = abc$$

- where a,b,c are the normalized values of the other incident crack edges
- m=max (a,b,c,q) ... the introduction of the quantity q ensures that type (0) is non-zero for small values of a.

- For example, choosing q=0.1, a vertex (a,b,c) = (0.5, 0.05, 0.05) is a type 1 vertex, while a vertex (0.3, 0.2, 0.2) is a type 3 vertex.

- Similar results can be obtained by simply counting the number of edges emanating from the vertex above a threshold value.

- Edge type is found as a simple concatenation of vertex types, and edge confidences are modified as follows:

Confidence increase: $c^{(k+1)}(e) = \min(1, c^{(k)}(e) + \delta)$

Confidence increase: $c^{(k+1)}(e) = \max(0, c^{(k)}(e) - \delta)$

- Edge relaxation, as described above, rapidly improves the initial edge labeling in the first few iterations.

- Unfortunately, it often slowly drifts giving worse results than expected after larger numbers of iterations.
- The reason for this strange behavior is in searching for the global maximum of the edge consistency criterion over all the image, which may not give locally optimal results.
- A solution is found in setting edge confidences to zero under a certain threshold, and to one over another threshold which increases the influence of original image data.

- Therefore, one additional step must be added to the edge confidence computation

  If $c^{(k+1)}(e) > T_1$ then assign $c^{(k+1)}(e)=1$

  If $c^{(k+1)}(e) < T_2$ then assign $c^{(k+1)}(e)=0$

- Where T_1 and T_2 are parameters controlling the edge relaxation convergence speed and resulting border accuracy.

- This method makes multiple labeling possible; the existence of two edges at different directions in one pixel may occur in corners, crosses, etc.

- The relaxation method can easily be implemented in parallel.

Border tracing

- If a region border is not known but regions have been defined in the image, borders can be uniquely detected.

- First, let us assume that the image with regions is either binary or that regions have been labeled.

- An **inner** region border is a subset of the region

- An **outer** border is not a subset of the region.

- The following algorithm covers inner boundary tracing in both 4-connectivity and 8-connectivity.

Algorithm: Inner Boundary tracing

1. Search the image from top left until a pixel of a new region is found; this pixel P_0 then has the minimum column value of all pixels of that region having the minimum row value. Pixel P_0 is a starting pixel of the region border. Define a variable dir which stores the direction of the previous move along the border from the previous border element to the current border element. Assign
   (a) $dir = 3$ if the border is detected in 4-connectivity (Fig 6(a))
   (b) $dir = 7$ if the border is detected in 8-connectivity (Fig 6(b))
2. Search the 3x3 neighborhood of the current pixel in an anti-clockwise direction, beginning the neighborhood search in the pixel positioned in the direction
   (a) $(dir + 3)$ mod 4 (Fig 6(c))
   (b) $(dir + 7)$ mod 8 if dir is even (Fig 6(d))
   (c) $(dir + 6)$ mod 8 if dir is odd (Fig 6(e))

   The first pixel found with the same value as the current pixel is a new boundary element P_n . Update the dir value.

3. If the current boundary element P_n is equal to the second border element P_1, and if the previous border element P_{n-1} is equal to P_0, stop. Otherwise repeat step 2.
4. The detected inner border is represented by pixels $P_0 \ldots\ldots P_{n-2}$.

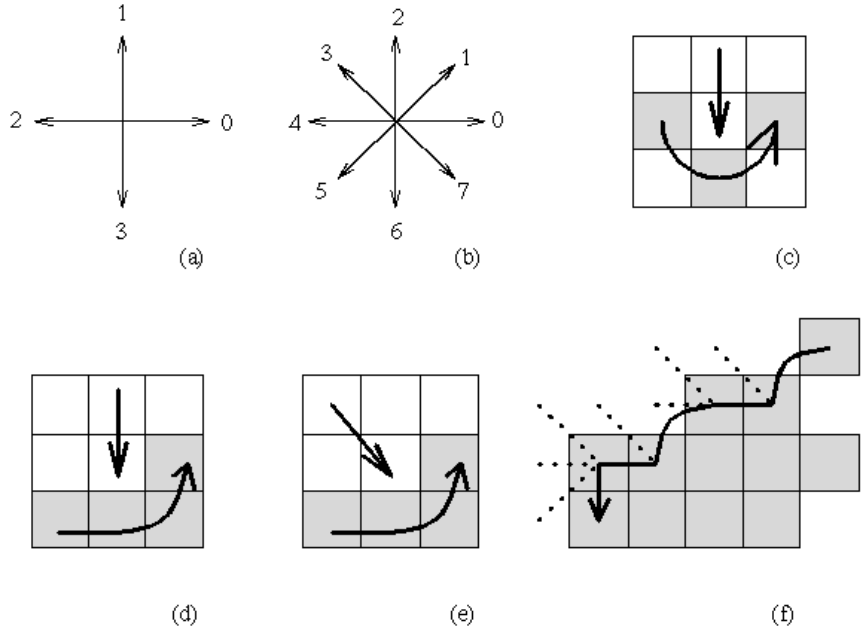


Fig 6:Inner boundary tracing (a)Direction notation,4-connectivity (b)8-connectivity (c)Pixel neighborhood search sequence in 4-connectivity (e)Search sequence in 8-connectivity (f)Boundary tracing in 8-connectivity

The above Algorithm works for all regions larger than one pixel.

- Looking for the border of a single-pixel region is a trivial problem.

- This algorithm is able to find region borders but does not find borders of region holes.
- To search for whole borders as well, the border must be traced starting in each region or whole border element if this element has never been a member of any border previously traced.

- Note that if objects are of unit width, more conditions must be added.

- If the goal is to detect an **outer region border**, the given algorithm may still be used based on 4-connectivity.

- Note that some border elements may be repeated in the outer border up to three times.

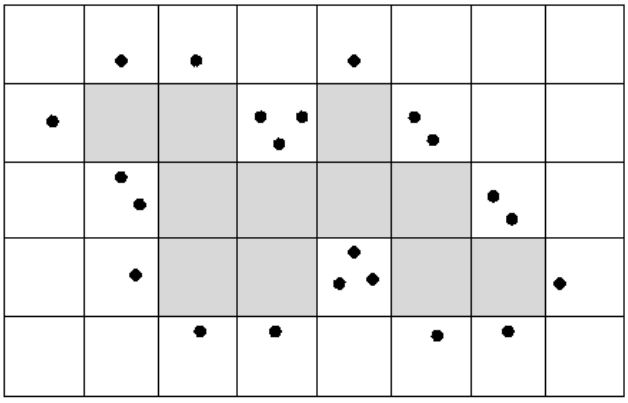


Fig 6: Outer boundary tracing

Algorithm: Outer boundary Tracing

1. Trace the inner region boundary in 4-connectivity until done.
2. The outer boundary consists of all non-region pixels that were tested during the search process; if some pixels were tested more than once, they are listed more than once in the boundary list.

- The inner border is always part of a region but the outer border never is.

- Therefore, if two regions are adjacent, they never have a common border, which causes difficulties in higher processing levels with region description, region merging, etc.

- The inter-pixel boundary extracted, for instance, from crack edges is common to adjacent borders; nevertheless, its position cannot be specified in pixel co-ordinates.

- Boundary properties better than those of outer borders may be found in **extended** borders.
    - single common border between adjacent regions
    - may be specified using standard pixel co-ordinates
    - the boundary shape is exactly equal to the inter-pixel shape but is shifted one half-pixel down and one half-pixel right

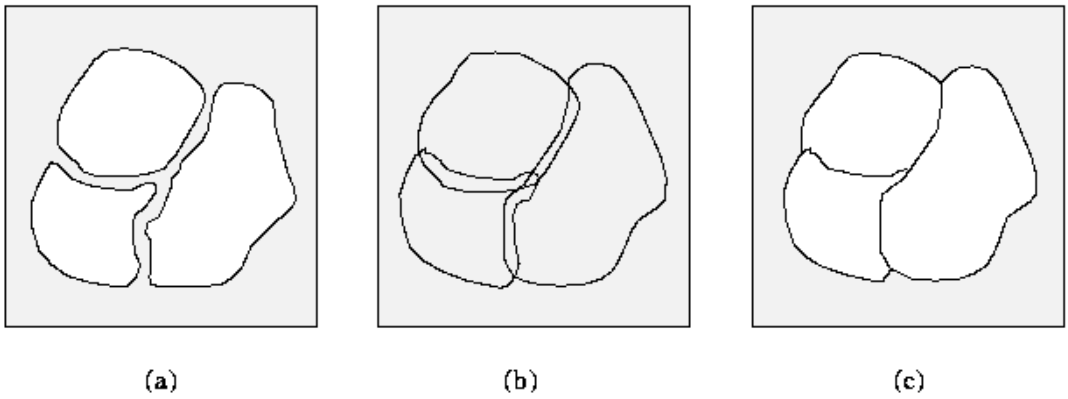


Fig 7: Boundary locations (a) Inner (b) Outer (c) Extended.

- The extended boundary is defined using 8-neighborhoods
- e.g.P_4(P) denotes the pixel immediately to the left of pixel P
- Four kinds of inner boundary pixels of a region R are defined; if Q denotes pixels outside the region R, then

LEFT pixel if P_4(P) ∈ Q

RIGHT pixel if P_0(P) ∈ Q

UPPER pixel if P_2(P) ∈ Q

LOWER pixel if P_6(P) ∈ Q

- Let LEFT(R), RIGHT(R), UPPER(R), LOWER(R) represents the corresponding subsets of R.
- The extended boundary EB is defined as a set of points P, P_0, P_6,P_7 satisfying the following conditions:

$$EB = \{P: P \in LEFT(R)\} \cup \{P: P \in UPPER(R)\} \cup$$
$$\{P_5(P): P \in LOWER(R)\} \cup \{P_6(P): P \in LEFT(R)\} \cup$$
$$\{P_0(P): P \in RIGHT(R)\} \cup \{P_7(P): P \in RIGHT(R)\}$$

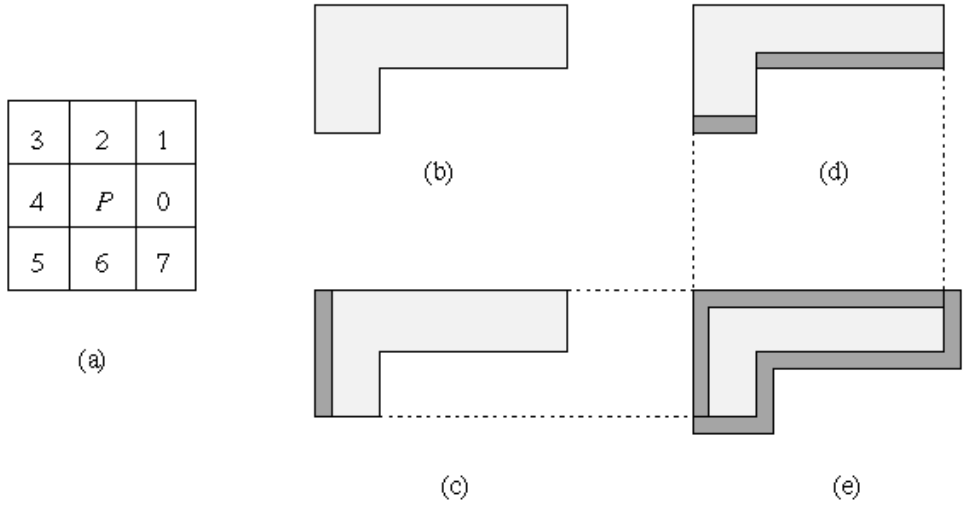


Fig 8: Extended boundary definition (a)Pixel coding scheme (b)Region R (c)LEFT(R) (d)LOWER(R) (e)Extended boundary.

- The extended boundary can easily be constructed from the outer boundary.
- Using an intuitive definition of RIGHT, LEFT, UPPER, and LOWER outer boundary points, the extended boundary may be obtained by shifting all the UPPER outer boundary points one pixel down and right, shifting all the LEFT outer boundary points one pixel to the right, and shifting all the RIGHT outer boundary points one pixel down. The LOWER outer boundary point positions remain unchanged.

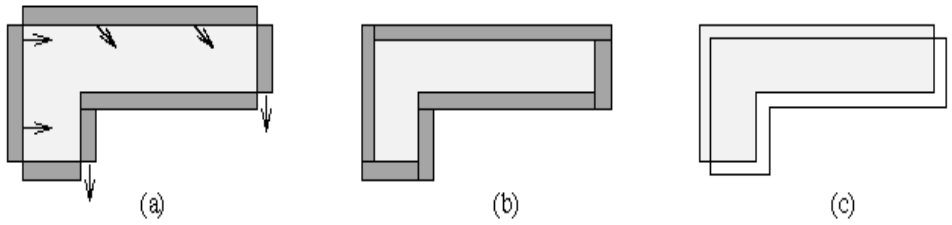


Fig 9: Constructing the extended boundary from outer boundary (a)Outer boundary (b)Extended boundary construction (c)Extended boundary has the same shape and size as the natural object boundary

- A more sophisticated approach is based on detecting common boundary segments between adjacent regions and vertex points in boundary segment connections.
- The detection process is based on a look-up table, which defines all 12 possible situations of the local configuration of 2 x 2 pixel windows, depending on the previous

detected direction of boundary, and on the status of window pixels which can be inside or outside a region.

Algorithm: Extended boundary tracing

1. Define a starting pixel of an extended boundary in a standard way (the first region pixel found in a left-to-right and top-to-bottom line-by-line image search).
2. The first move along the traced boundary from the starting pixel is in direction *dir*=6 (down), corresponding to the situation *(i)* in Fig 10
3. Trace the extended boundary using the look-up table in Fig 10 until a closed extended border results.

- Note that there is no hole-border tracing included in the algorithm.
- The holes are considered separate regions and therefore the borders between the region and its hole are traced as a border of the hole.

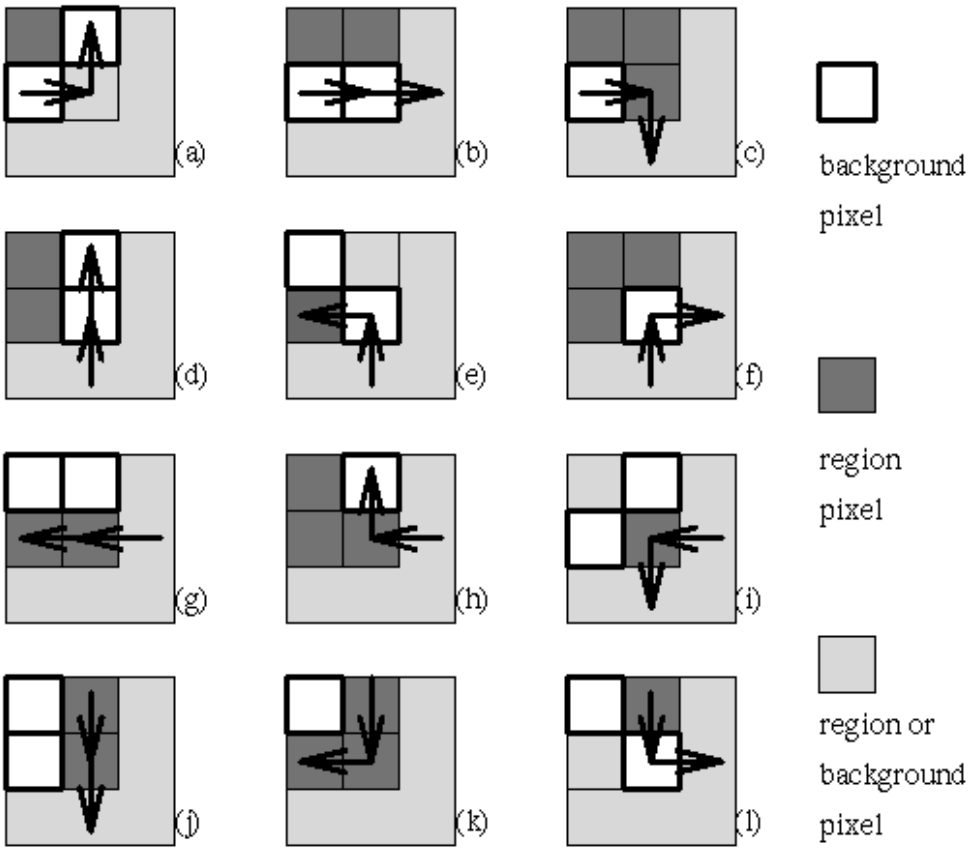


Fig 10: look up table defining all 12 possible situations that can appear during extended border tracing.

- The look-up table approach makes the tracing more efficient than conventional methods and makes parallel implementation possible.

- In addition to extended boundary tracing, it provides a description of each boundary segment in chain code form together with information about vertices.

- This method is very suitable for representing borders in higher level segmentation approaches including methods that integrate edge-based and region-based segmentation results.

- Moreover, in the conventional approaches, each border between two regions must be traced twice. The algorithm can trace each boundary segment only once storing the information about what has already been done in double-linked lists.

- A more difficult situation is encountered if the borders are traced in grey level images where regions have not yet been defined.
- The border is represented by a simple path of high-gradient pixels in the image.
- Border tracing should be started in a pixel with a high probability of being a border element, and then border construction is based on the idea of adding the next elements which are in the most probable direction.
- To find the following border elements, edge gradient magnitudes and directions are usually computed in pixels of probable border continuation.

Border detection as graph searching

- Whenever additional knowledge is available for boundary detection, it should be used - e.g., known approximate starting point and ending point of the border
- Even some relatively weak additional requirements such as smoothness, low curvature, etc. may be included as prior knowledge.

- A graph is a general structure consisting of a set of nodes n_i and arcs between the nodes $[n_i, n_j]$. We consider oriented and numerically weighted arcs, these weights being called **costs**.

- The border detection process is transformed into a search for the optimal path in the weighted graph.
- Assume that both edge magnitude and edge direction information is available in an edge image.

- Figure 11 (a) shows an image of edge directions, with only significant edges according to their magnitudes listed.

- Figure 11 (b) shows an oriented graph constructed in accordance with the presented principles.

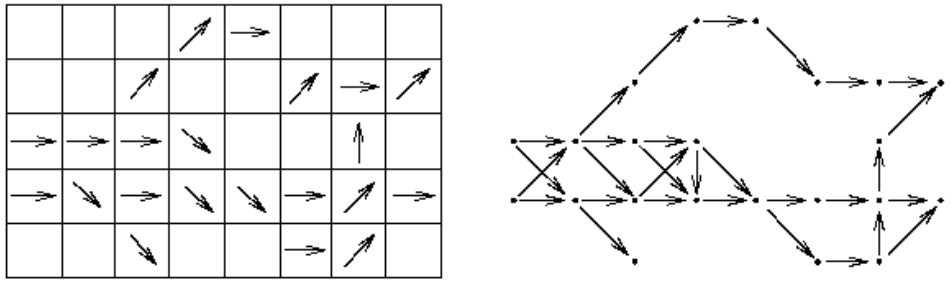


Fig 11: Graph representation of an edge image. (a)Edge directions of pixels with above-threshold edge magnitudes. (b) Corresponding graph.

To use graph search for region border detection, a method of oriented weighted-graph expansion must first be defined.

Graph Search Example

- A cost function $f(x_i)$ must also be defined that is a cost estimate of the path between nodes n_A and n_B (pixels $x_A$and x_B) which goes through an intermediate node n_i (pixel x_i).
- The cost function $f(x_i)$ typically consists of two components; an estimate of the path cost between the starting border element x_A and x_i, and an estimate of the path cost between $x_i$and the end border element x_B.
- The cost of the path from the starting point to the node n_i is usually a sum of costs associated with the arcs or nodes that are in the path.
- The cost function must be separable and monotonic with respect to the path length, and therefore the local costs associated with arcs are required to be non-negative.

Algorithm: Heuristic Graph search

1. Expand the starting node n_A and put all its successors into an OPEN list with pointers back to the starting node n_A. Evaluate the cost function *f* for each expanded node.
2. If the OPEN list is empty, fail. Determine the node n_i from the OPEN list with the lowest associated cost $f(n_i)$ and remove it. If n_i=n_B then trace back through the pointers to find the optimum path and stop.

3. If the option to stop was not taken in step 2, expand the specified node n_i, and put its successors on the OPEN list with pointers back to n_i. Compute their costs *f*. Go to step 2.

- If no additional requirements are set on the graph construction and search, this process can easily result in an infinite loop.

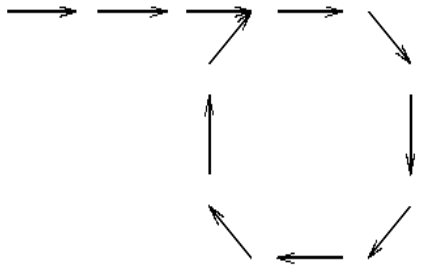


Fig 12: Example of following a closed loop in image data.

- To prevent this behavior, no node expansion is allowed that puts a node on the OPEN list if this node has already been visited, and put on the OPEN list in the past.
- A simple solution to the loop problem is not to allow searching in a backward direction.
- It may be possible to straighten the processed image (and the corresponding graph).

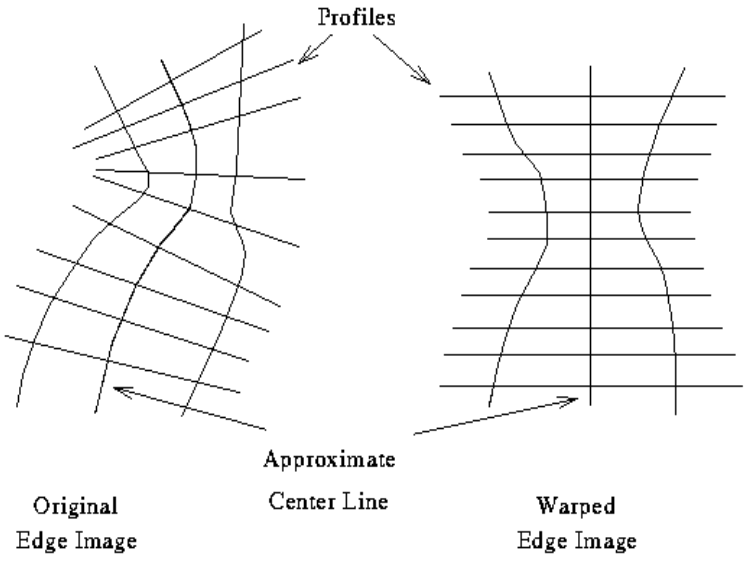


Fig 13: Geometric warping produces a straightened image: The graph constructed requires searches in one direction only (e.g., Top Down).

- The estimate of the cost of the path from the current node n_i to the end node n_E has a substantial influence on the search behavior.
- If this estimate $\sim h(n_i)$ of the true cost $h(n_i)$ is not considered, so $\sim h(n_i) = 0$ no heuristic is included in the algorithm and a breadth-first search is done.

- The detected path will always be optimal according to the criterion used, the minimum cost path will always be found.

- Applying heuristics, the detected cost does not always have to be optimal but the search can often be much faster.

- The minimum cost path result can be guaranteed if $\sim h(n_i) <= h(n_i)$ and if the true cost of any part of the path $c(n_p, n_q)$ is larger than the estimated cost of this part $\sim c(n_p, n_q)$.
- The closer the estimate $\sim h(n_i)$ is to $h(n_i)$, the lower the number of nodes expanded in the search.
- The problem is that the exact cost of the path from the node n_i to the end node n_E is not known beforehand.

- In some applications, it may be more important to get a quick rather than an optimal solution. Choosing $\sim h(n_i) > h(n_i)$, optimality is not guaranteed but the number of expanded nodes will typically be smaller because the search can be stopped before the optimum is found.

- If $\sim h(n_i) = 0$, the algorithm produces a minimum-cost search.
- If $\sim h(n_i) <= h(n_i)$, the search will produce the minimum-cost path if and only if the conditions presented earlier are satisfied.
- If $\sim h(n_i) = h(n_i)$, the search will always produce the minimum-cost path with a minimum number of expanded nodes.
- If $\sim h(n_i) > h(n_i)$, the algorithm may run faster, but the minimum-cost result is not guaranteed.
- The better the estimate of $h(n)$, the smaller the number of nodes that must be expanded.

- A crucial question is how to choose the evaluation cost functions for graph-search border detection.

- Some generally applicable cost functions are
- Strength of edges forming a border

$$\left(max_{image}\ s(X_k)\right) - s(X_i)$$

- where the maximum edge strength is obtained from all pixels in the image.
- Border curvature

$$DIF\,(\emptyset(X_i) - \emptyset(X_j))$$

- where DIF is some suitable function evaluating the difference in edge directions in two consecutive border elements.
- Proximity to an approximate border location

$$dist(X_i, approximate - boundary)$$

- Estimates of the distance to the goal (end-point)

$$\hat{h}(X_i) = dist\,(X_i, X_g)$$

- Since the range of border detection applications is quite wide, the cost functions described may need some modification to be relevant to a particular task.

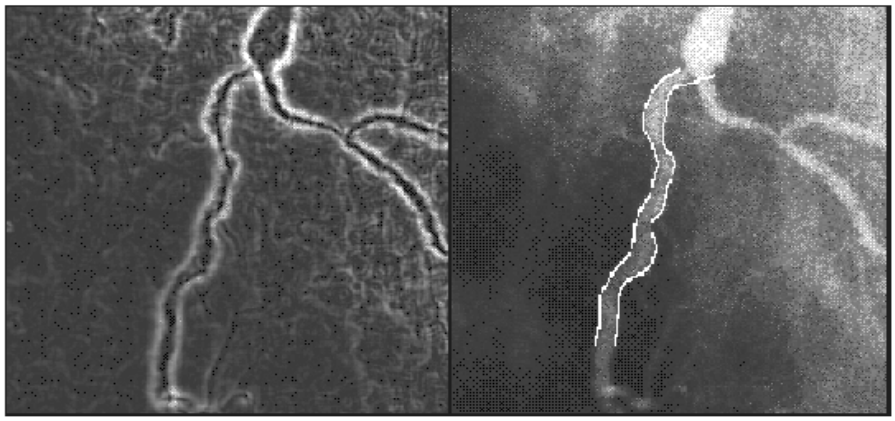


Fig 14: Graph search applied to coronary vessel border detection. (a) Edge image (b) Determined vessel borders.

- Graph searching techniques offer a convenient way to ensure global optimality of the detected contour.
- The detection of closed structure contours would involve geometrically transforming the image using a polar to rectangular co-ordinate transformation in order to `straighten' the contour.

- Searching for all the borders in the image without knowledge of the start and end-points is more complex.
- Edge chains may be constructed by applying a bidirectional heuristic search in which half of each 8-neighborhood expanded node is considered as lying in front of the edge, the second half as lying behind the edge.

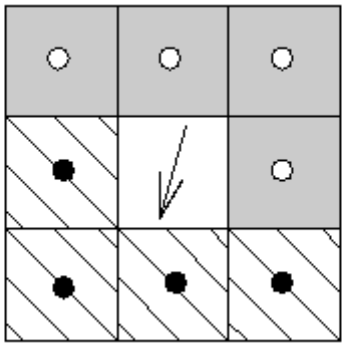


Fig 15: Bidirectional heuristic search: Edge predecessors(marked ∩) and successors(marked ●)

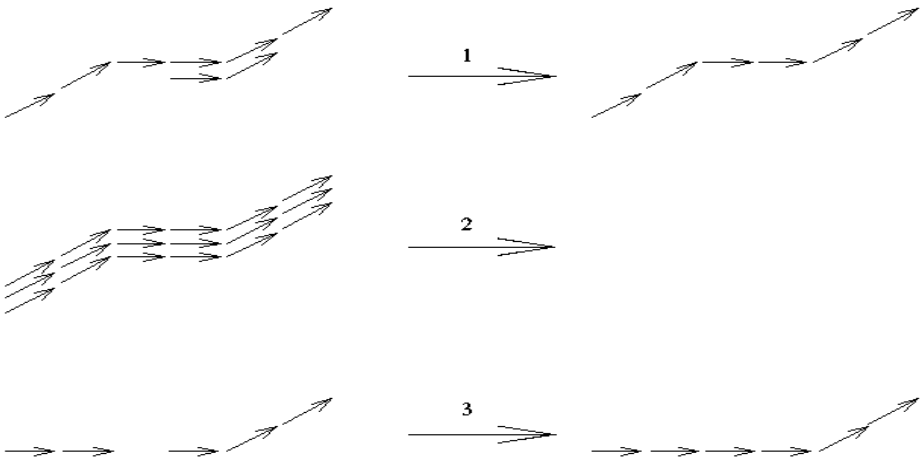


Fig 16: Rules for edge chain modification; note that edge responses resulting from continuous changes of illumination are removed.

Border detection as dynamic programming

- Dynamic programming is an optimization method based on the **principle of optimality**.
- It searches for optima of functions in which not all variables are simultaneously interrelated.

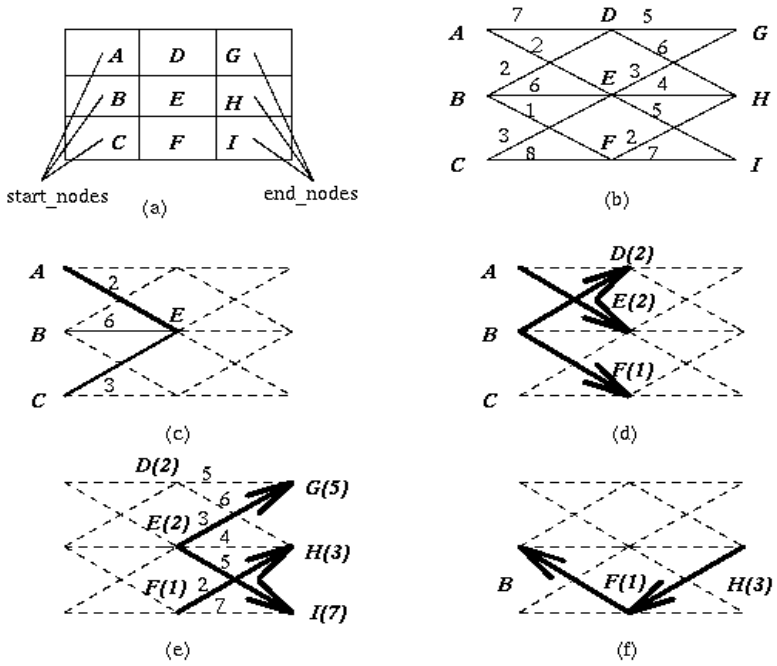


Fig 17: Edge following as dynamic programming. (a) Edge image. (b) Corresponding graph, partial costs assigned. (c) Possible paths from any start point to E,A-E is optimal. (d)Optimal partial paths to nodes D,E,F. (e) Optimal partial paths to nodes G,H,I. (f) Back-tracking from H defines the optimal boundary.

- The main idea of the principle of optimality is:
  - **Whatever the path to the node E was, there exists an optimal path between E and the end-point.**
  - In other words, **if the optimal path start-point -- end-point goes through E then both its parts start-point -- E and E -- end-point is also optimal**.

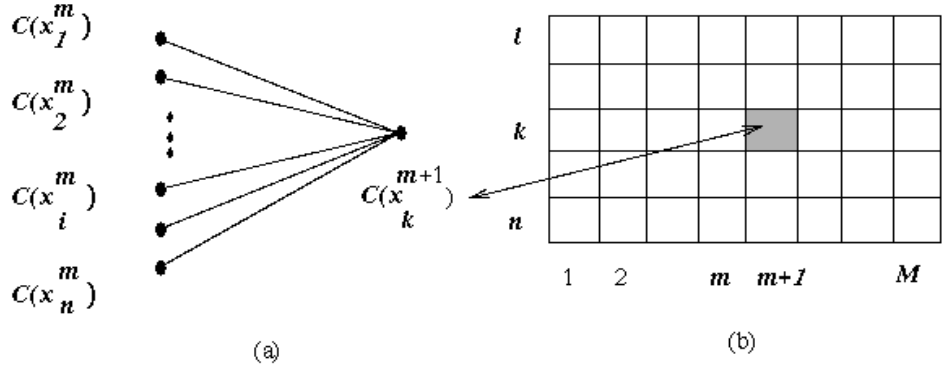


Fig 18: Dynamic programming: (a) One step of the cost calculation; (b) Graph layers, node notation.

- If the graph has more layers, the process is repeated until one of the end-points is reached.

- The complete graph must be constructed to apply dynamic programming, and this may follow general rules given in the previous section.
- The objective function must be separable and monotonic (as for the A-algorithm); evaluation functions presented in the previous section may also be appropriate for dynamic programming.

Algorithm: Boundary tracing as dynamic programming

1. Specify initial costs $C(x_i^1)$ of all nodes in the first graph layer, *i=1,…,n* and partial path costs $g^m(i,k)$, m=1,…,M-1.
2. Repeat step 3 for all m=1… M-1.
3. Repeat step 4 for all nodes *k=1,…,n* in the graph layer m.
4. Let

$$C(x_k^{m+1}) = \min_{i=-1,0,1}(C(x_{k+i}^m) + g^m(i,k)).$$

Set pointer from node x_k^{m+1} back to node x_i^{m*} ; where * denotes the optimal predecessor.

Find an optimal node x_k^{M*} in the last graph layer M and obtain an optimal path by back-tracking through the pointers from x_k^{M*} to x_i^{1*} .

Heuristic Graph Search vs. Dynamic Programming

- It has been shown that heuristic search may be more efficient than dynamic programming for finding a path between two nodes in a graph if a good heuristic is available.
- A-algorithm based graph search does not require explicit definition of the graph.
- Dynamic programming presents an efficient way of simultaneously searching for optimal paths from multiple starting and ending points.
- If these points are not known, dynamic programming is probably a better choice, especially if computation of the partial costs $g^m(i,k)$ is simple.

- Nevertheless, which approach is more efficient for a particular problem depends on evaluation functions and on the quality of heuristics for an A-algorithm. A

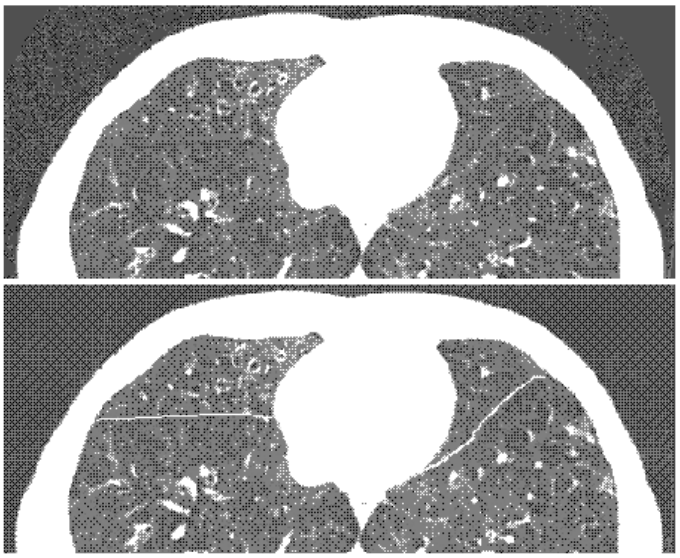


Fig 19: Detection of pulmonary fissures using dynamic programming. Top: sub-region of an original cross-sectional X-ray CT image of a human lung. Bottom: Detected fissures shown in white.

- **Live wire** combines automated border detection with manual definition of the boundary start-point and interactive positioning of the end-point.

- In dynamic programming, the graph that is searched is always completely constructed at the beginning of the search process. Therefore, interactive positioning of the end-point invokes no time-consuming recreation of the graph as would be the case in heuristic graph searching.

- Thus, after construction of the complete graph and associated node costs, optimal borders connecting the fixed start-point and the interactively changing end-point can be determined in real time.

- In the case of large or more complicated regions, the complete region border is usually constructed from several border segments.

- In the **live lane** approach, an operator defines a region of interest by approximately tracing the border by moving a square window.

- The size of the window is either preselected or is adaptively defined from the speed and acceleration of the manual tracing. When the border is of high quality, the manual tracing is fast and the live lane method is essentially identical to the live wire method applied to a sequence of rectangular windows. If the border is less obvious, manual tracing is usually slower and the window size adaptively decreases.

- If the window size reduces to a single pixel, the method degenerates to manual tracing.

- A flexible method results that combines the speed of automated border detection with the robustness of manual border detection whenever needed.

- Since the graph is only constructed using the image portion comparable in size to the size of the moving window, the computational demands of the live lane method are much lower than that of the live wire method.

- Several additional features of the two **live** methods are worth mentioning.
    - Design of border detection cost functions often requires substantial experience and experimentation.
    - To facilitate the method's usage by non-experts, an automated approach has been developed that determines optimal border features from examples of the correct borders.
    - The resultant optimal cost function is specifically designed for a particular application and can be conveniently obtained by simply presenting a small number of example border segments during the method's training stage.

Hough transforms

- If an image consists of objects with known shape and size, segmentation can be viewed as a problem of finding this object within an image.
- One powerful global method for detecting edges is called the ***Hough transform***.

- Consider an example of circle detection.
- Search the dark image pixels by this potential center points of the circle can be determined which forms a circle with radius r [Fig 20(b)].
- After the construction of potential circle for all dark pixels of an image the frequency is determined. The true center of the circle is represented by the pixel with the highest frequency [Fig 20:(c)].
- Fig 20(d) presents that the Hough transform can apply to the images with incomplete information or additional structures and noise.

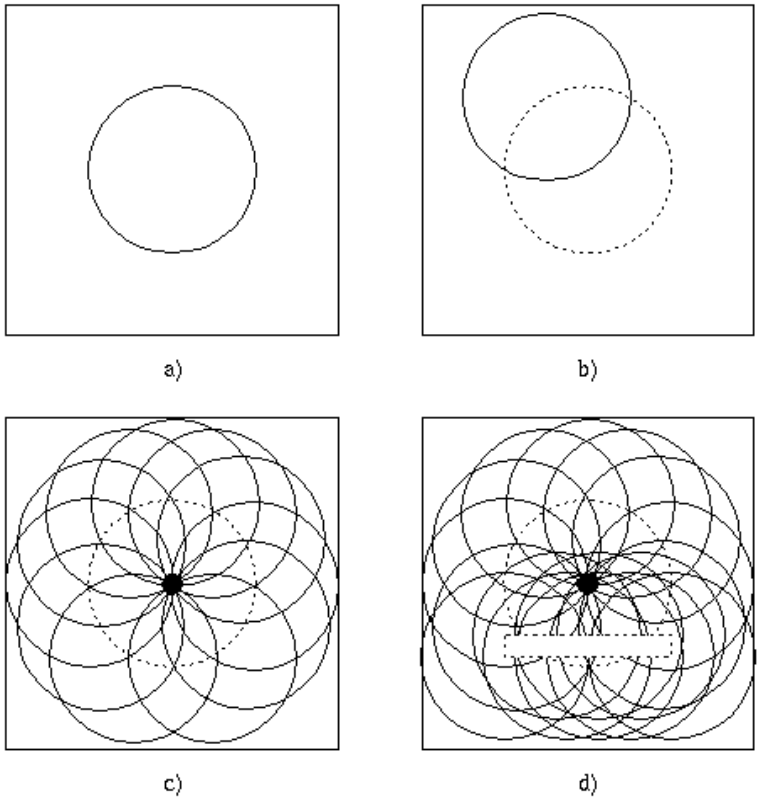


Fig 20: Hough transform(circle detection) (a)Original image (b)Original image of dark circle(radius r) (c)The highest frequency pixel represents the centre of the circle (d)Hough transform detects the circle 9in incomplete and overlapping structures

- The original Hough transform was designed to detect straight lines and curves
- A big advantage of this approach is robustness of segmentation results; that is, segmentation is not too sensitive to imperfect data or noise.

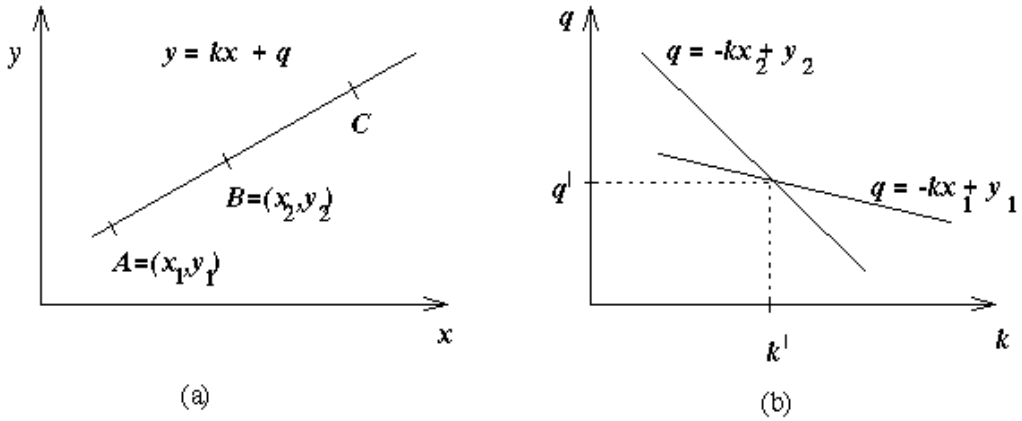


Fig 21: Hough transform principles (a) Image space (b) *k, q* parameter space.

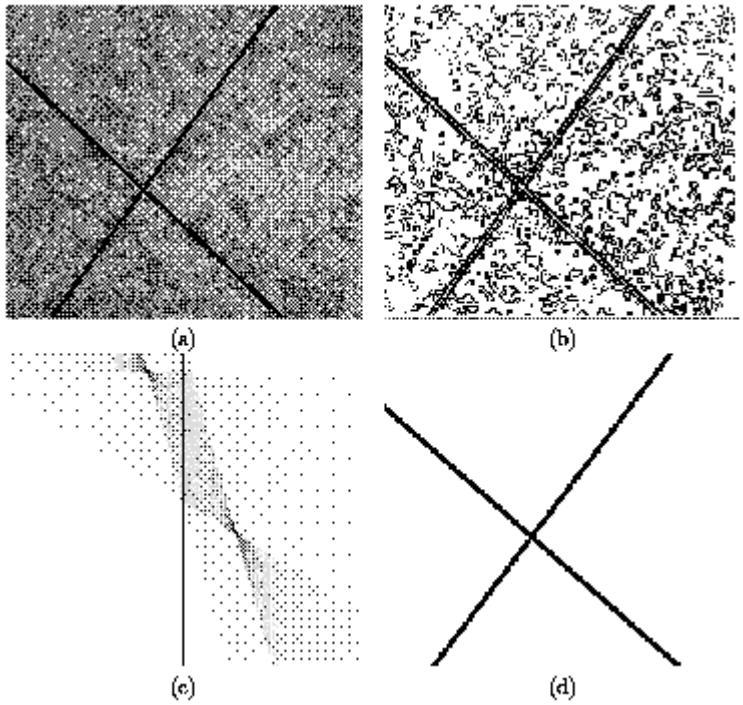


Fig 22:Hough transform-Line detection (a)Original image (b)Edge image (c) Parameter space (d) Detected lines

This means that any straight line in the image is represented by a single point in the k,q parameter space and any part of this straight line is transformed into the same point.

- The main idea of line detection is to determine all the possible line pixels in the image, to transform all lines that can go through these pixels into corresponding points in the parameter space, and to detect the points (a,b) in the parameter space that frequently resulted from the Hough transform of lines y=ax+b in the image.

- Detection of all possible line pixels in the image may be achieved by applying an edge detector to the image

- Then, all pixels with edge magnitude exceeding some threshold can be considered possible line pixels.

- In the most general case, nothing is known about lines in the image, and therefore lines of any direction may go through any of the edge pixels. In reality, the number of these lines is infinite, however, for practical purposes, only a limited number of line directions may be considered.

- The possible directions of lines define a discretization of the parameter k.
- Similarly, the parameter q is sampled into a limited number of values.

- The parameter space is not continuous any more, but rather is represented by a rectangular structure of cells. This array of cells is called the **accumulator array** A, whose elements are **accumulator cells** A(k, q).

- For each edge pixel, parameters k, q is determined which represent lines of allowed directions going through this pixel. For each such line, the values of line parameters k, q are used to increase the value of the accumulator cell A(k, q).

- Clearly, if a line represented by an equation y=ax+b is present in the image, the value of the accumulator cell A(a,b) will be increased many times -- as many times as the line y=ax+b is detected as a line possibly going through any of the edge pixels.

- Lines existing in the image may be detected as high-valued accumulator cells in the accumulator array, and the parameters of the detected line are specified by the accumulator array co-ordinates.

- As a result, line detection in the image is transformed to detection of local maxima in the accumulator space.

- The parametric equation of the line y=kx+q is appropriate only for explanation of the Hough transform principles -- it causes difficulties in vertical line detection (k -> infinity) and in nonlinear discretization of the parameter k.

- .The parameterizations of the lines used in this demonstration are given by the polar form of a line.

- If a line is represented as

$$s = x \cos\theta + y \sin\theta$$

the Hough transform does not suffer from these limitations.

- Again, the straight line is transformed to a single point

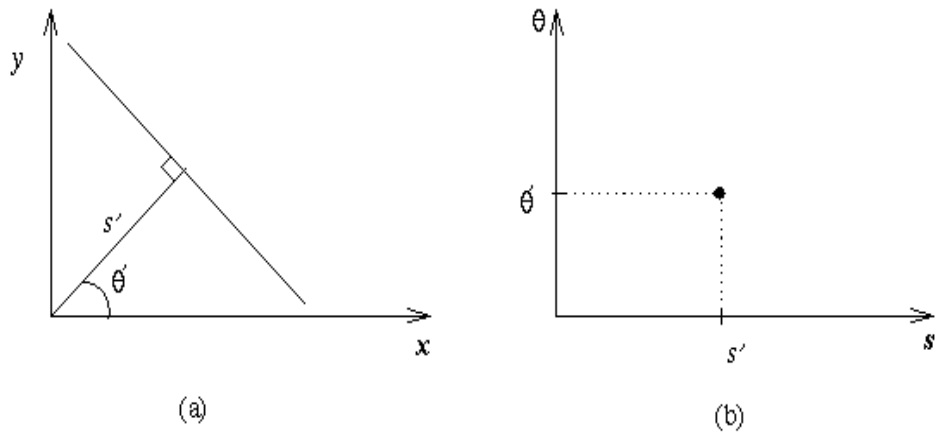


Fig 23: Hough transform in s, θ space (a) Straight line in image space (b) s, θ parameter space.

- Discretization of the parameter space is an important part of this approach.
- Also, detecting the local maxima in the accumulator array is a non-trivial problem.
- In reality, the resulting discrete parameter space usually has more than one local maximum per line existing in the image, and smoothing the discrete parameter space may be a solution.

- Generalization to more complex curves that can be described by an analytic equation is straightforward.
- Consider an arbitrary curve represented by an equation f(x, a) =0, where {a} is the vector of curve parameters.

Algorithm: Curve detection using the Hough transform

1. Quantize parameter space within the limits of parameters **a.** The dimensionality *n* of the parameter space is given by the number of parameters of the vector **a.**
2. Form an *n*-dimensional accumulator array A(**a**) with structure matching the quantization of parameter space; set all elements to zero.
3. For each image point (x_1, x_2) in the appropriately thresholded gradient image, increase all accumulator cells A(**a**) if *f(**x**,**a**)*=0

   $$A(\mathbf{a}) = A(\mathbf{a}) + \Delta A$$

   For all **a** inside the limits used in step 1.

4. Local maxima in the accumulator array A(a) correspond to realizations of curves *f(**x**,**a**)* that are present in the original image.

If we are looking for circles, the analytic expression f(x,a) of the desired curve is

$$(x_1 - a)^2 + (x_2 - b)^2 = r^2$$

- where the circle has center (a,b) and radius r.

- Therefore, the accumulator data structure must be three-dimensional.

- Even though the Hough transform is a very powerful technique for curve detection, exponential growth of the accumulator data structure with the increase of the number of curve parameters restricts its practical usability to curves with few parameters.

- If prior information about edge directions is used, computational demands can be decreased significantly.
- Consider the case of searching the circular boundary of a dark region, letting the circle have a constant radius r=R for simplicity.
- Without using edge direction information, all accumulator cells A(a,b) are incremented in the parameter space if the corresponding point (a,b) is on a circle with center x.

- With knowledge of direction, only a small number of the accumulator cells need be incremented. For example, if edge directions are quantized into 8 possible values, only one eighth of the circle need take part in incrementing of accumulator cells.
- Using edge directions, candidates for parameters a and b can be identified from the following formulae:

$$a = x_1 - R\cos(\psi(X))$$
$$b = x_2 - R\sin(\psi(X))$$
$$\psi(X)\epsilon[\phi(X) - \Delta\phi, \phi(X) + \Delta\phi]$$

- where $\phi(X)$ refers to the edge direction in pixel x and $\Delta\phi$ is the maximum anticipated edge direction error.
- Another heuristic that has a beneficial influence on the curve search is to weight the contributions to accumulator cells A(a) by the edge magnitude in pixel x

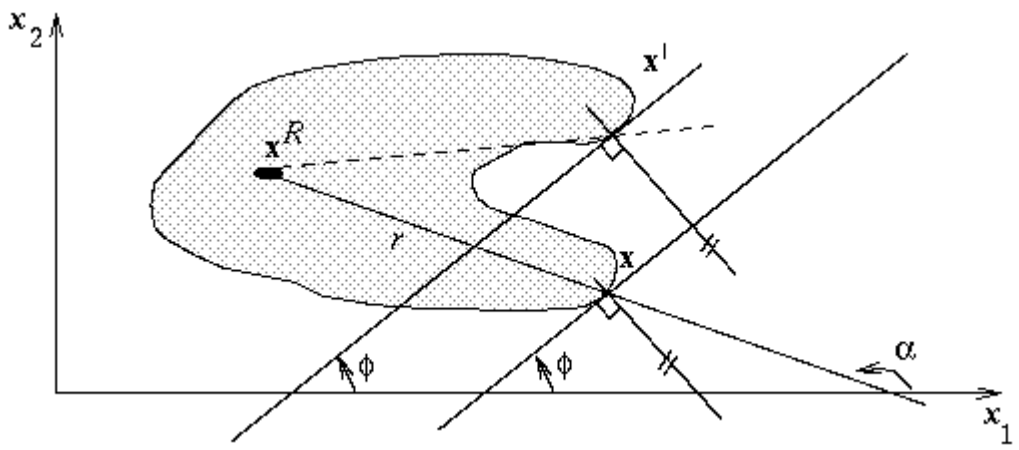


Fig 24: Principles of the generalized Hough transform: geometry of **R**-table construction.

R-Table

$\phi_1 \quad (r_1^1, \alpha_1^1), (r_1^2, \alpha_1^2), \ldots\ldots, (r_1^{n_1}, \alpha_1^{n_1})$

$\phi_2 \quad (r_2^1, \alpha_2^1), (r_2^2, \alpha_2^2), \ldots\ldots, (r_2^{n_2}, \alpha_2^{n_2})$

$\phi_3 \quad (r_3^1, \alpha_3^1), (r_3^2, \alpha_3^2), \ldots\ldots, (r_3^{n_3}, \alpha_3^{n_3})$

…… ……

$\phi_k \quad (r_k^1, \alpha_k^1), (r_k^2, \alpha_k^2), \ldots\ldots, (r_k^{n_k}, \alpha_k^{n_k})$

Algorithm: Generalized Hough transform

1. Construct an R-table description of the desired object.
2. Form a data structure A that represents the potential reference points $A(x_1, x_2, S, \tau)$

   Set all accumulators cell values $A(x_1, x_2, S, \tau)$ to zero.

3. For each pixel (x_1, x_2) in a threshold gradient image, determine the edge direction $\Phi(X)$; find all potential reference points X^R and increase all

$A(X^R, S, \tau) = A(X^R, S, \tau) + \Delta A$

For all possible values of rotation and size change

$$x_1^R = x_1 + r(\phi + \tau)S\cos(\alpha(\phi + \tau))$$

$$x_2^R = x_2 + r(\phi + \tau)S\sin(\alpha(\phi + \tau))$$

4. The location of suitable regions is given by local maxima in the A data structure.

Border detection using border location information

Boundary is determined in an image as the location of significant edges positioned close to an assumed border if the edge directions of these significant edges match the assumed boundary direction. The new border pixels are sought in directions perpendicular to the assumed border (Fig 25). If a large number of border elements satisfying the given conditions is found , an approximate curve is computed based on these pixels, and a new, more accurate , border results.

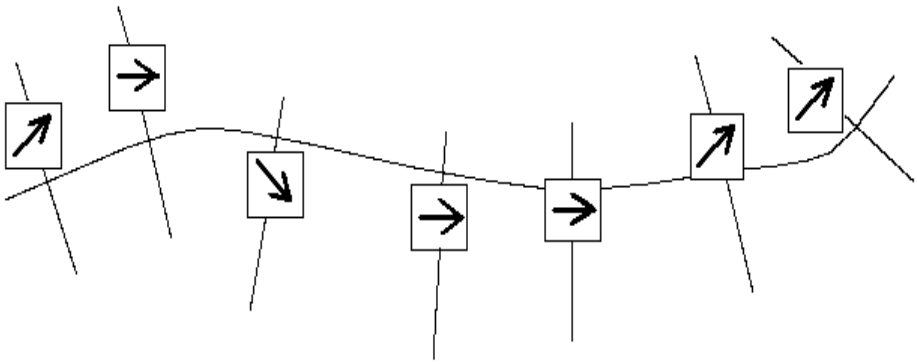


Fig 25: A priori information about boundary location

Another possibility is based on prior knowledge of end points – this approach assumes low image noise and relatively straight boundaries. This process iteratively partitions the border and searches for the strongest edge located on perpendiculars to the line connecting end points of each partition; perpendiculars are located at the center of the connecting straight line (Fig 26). The strongest significant edge located on the perpendicular that is close to the straight line connecting the end points of the current partition is accepted as a new border element. The iteration process is then repeated.

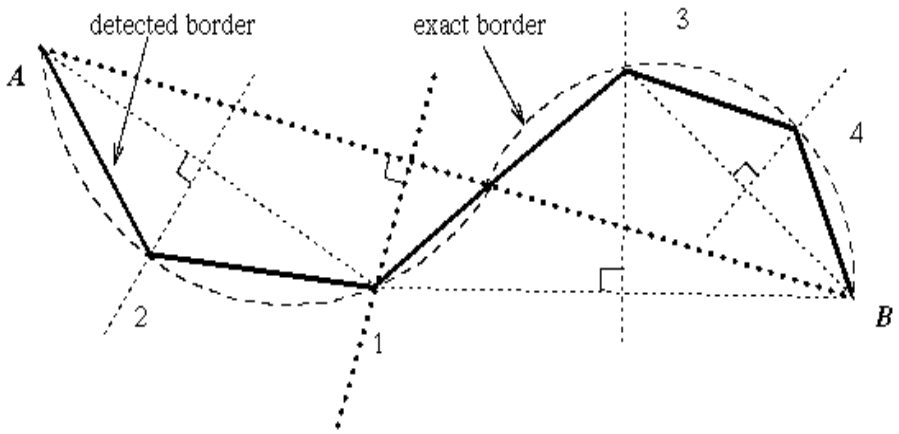


Fig 26: Divide-and-conquer iterative border detection; numbers show the sequence of division steps

Region construction from borders

Algorithm: Region forming from partial borders

1. For each border pixel **x**, search for an opposite edge pixel within a distance not exceeding a given maximum ***M***. If an opposite edge pixel is not found, process the next border pixel in the image. If an opposite edge pixel is found, mark each pixel on the connecting straight line as a potential region member.
2. Compute the number of markers for each pixel in the image (the number of markers tells how often a pixel was on a connecting line between opposite edge pixels). Let b(**x**) be the number of markers for the pixel **x**.
3. The weighted number of markers B(**x**) is then determined as follows:

   B(**x**) =0.0 for b(**x**) =0,
   B(**x**) =0.1 for b(**x**) =1,
   B(**x**) =0.2 for b(**x**) =2,
   B(**x**) =0.5 for b(**x**) =3,
   B(**x**) =1.0 for b(**x**) =>3.

   The confidence that a pixel x is a member of a region is given as the sum $\sum_i B(x_i)$ in a 3x3 neighborhood of the pixel x. If the confidence that a pixel x is a region member is one or larger, then pixel x is marked as a region pixel, otherwise it is marked as a background pixel.

This method allows the construction of bright regions on a dark background as well as dark regions on a bright background by taking either of the two options in the search for opposite edge pixels —step 1.

Search orientation depends on whether relatively dark or bright regions are constructed. If $\emptyset(X)$ and $\emptyset(Y)$ are directions of edges, the condition that must be satisfied for **X** and **Y** to be opposite is

$$\frac{\pi}{2} < \left|\left(\emptyset(X) - \emptyset(Y)\right) mod(2\pi)\right| < \frac{3\pi}{2}$$

Region-based Segmentation

- Region based segmentation is to partition an image into regions.
- It is easy to construct regions from their borders, and it is easy to detect borders of existing regions.

Basic Formulation

- Let R represent the entire image region. We may view segmentation as a process that partitions R into n sub regions, $R_1, R_2, \ldots\ldots, R_n$, such that

  (a) $\cup_{i=1}^{n} R_i = R$

  (b) R_i is a connected region, *i=1,2,. . . ., n.*

  (c) $R_i \cap R_j = \emptyset$ for all *i* and *j, i≠j.*

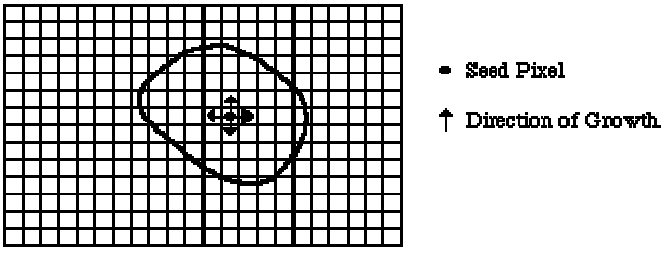
  (d) $P(R_i)$= TRUE for *i=1,2,. . . . ,n.*

  (e) $P(R_i \cup R_j)$=FALSE for *i≠j.*

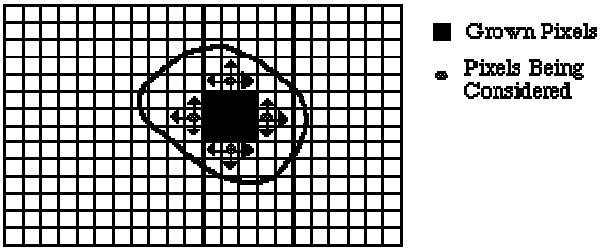
- Here $P(R_i)$ is a logical predicate defined over the points in set R_i and $\emptyset$ is the null set.
- Condition (a) indicates segmentation must be complete.
- (b) Requires that points in a region must be connected in some predefined sense
- (c) Indicates that the region must be disjoint.
- (d)Deals with the properties that must be satisfied by the pixels in a segmented region [$P(R_i)$= TRUE if all pixels in R_i have same gray level].
- (e) Indicates that regions R_i and R_j are different.

Region Growing

- Region growing approaches is the opposite of the split and merge approach:
- An initial set of small areas are iteratively merged according to similarity constraints. Start by choosing an arbitrary ***seed pixel*** and compare it with neighboring pixels.
- Region is grown from the seed pixel by adding in neighboring pixels that are similar, increasing the size of the region.
- When the growth of one region stops we simply choose another seed pixel which does not yet belong to any region and start again.
- This whole process is continued until all pixels belong to some region.
- Region growing methods often give very good segmentations that correspond well to the observed edges.



(a) Start of Growing a Region



(b) Growing Process After a Few Iterations

- However starting with a particular seed pixel and letting this region grow completely before trying other seeds biases the segmentation in favor of the regions which are segmented first.

- This can have several undesirable effects: Current region dominates the growth process ambiguities around edges of adjacent regions may not be resolved correctly.
- Different choices of seeds may give different segmentation results. Problems can occur if the (arbitrarily chosen) seed point lies on an edge.
- To counter the above problems, *simultaneous region growing* techniques have been developed.
- Similarities of neighboring regions are taken into account in the growing process.
- No single region is allowed to completely dominate the proceedings.
- A number of regions are allowed to grow at the same time.
- Similar regions will gradually merge into expanding regions. Control of these methods may be quite complicated but efficient methods have been developed.
- Easy and efficient to implement on parallel computers.

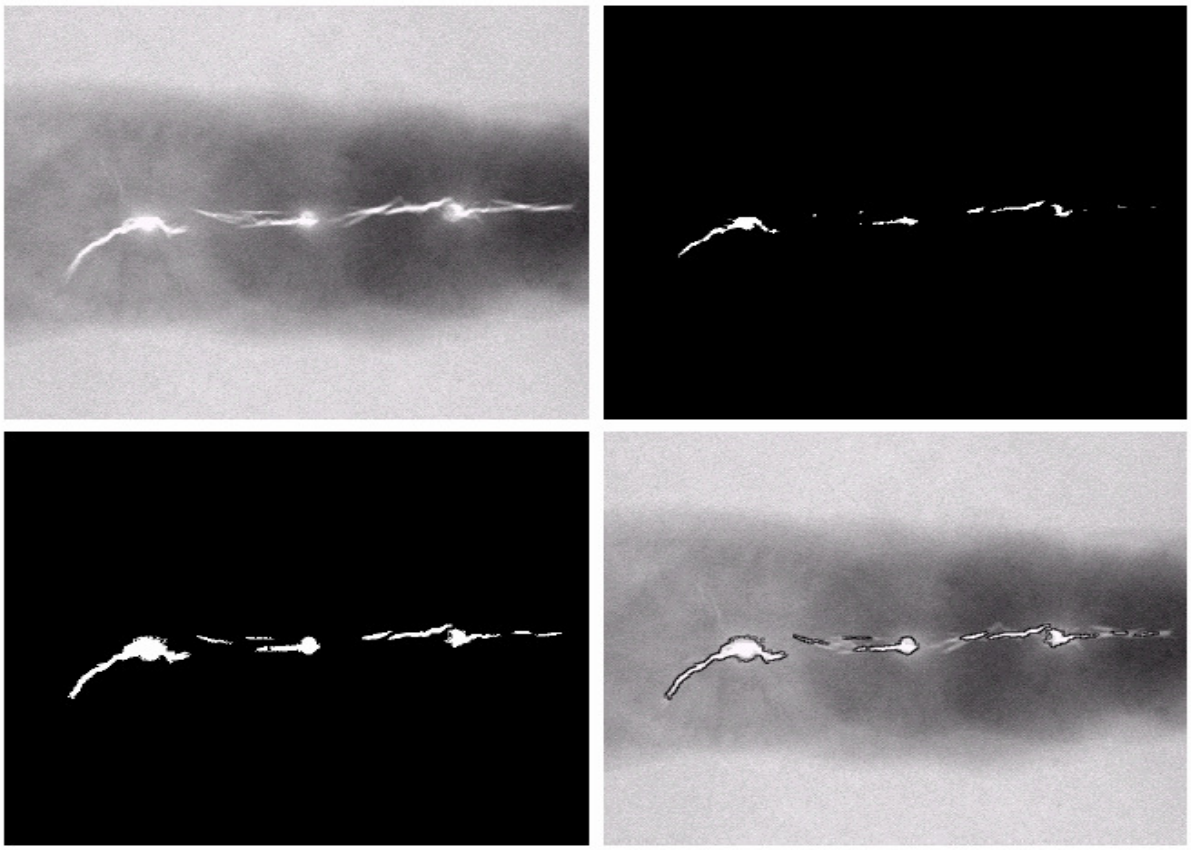


Fig 1(a)Image showing defective welds (b)Seed points (c)Result of region growing (d)Boundaries of segmented defective welds(in black)

Region Merging

- Region growing is a procedure that groups pixels or sub regions into larger regions based on predefined criteria.
- Here each pixel represents a single region.

Algorithm: Region merging(outline)

1. Define some starting method to segment the image into many small regions satisfying condition $P(R_i)$= TRUE for *i=1,2,. . . . , n.*

2. Define a criterion for merging two adjacent regions.

3. Merge all adjacent regions satisfying the merging criterion. If no two regions can be merged for the condition $P(R_i \cup R_j)$=FALSE for *i≠j,* stop

- Regions are parts of the image that can be sequentially merged into larger regions if they satisfy the conditions [$P(R_i)$= TRUE for *i=1, 2 . . . , n.* and $P(R_i \cup R_j)$=FALSE for *i≠j*] .
- The result of region merging usually depends on the order in which regions are merged, meaning that segmentation results will probably differ if segmentation begins, for instance, in the upper left or lower right corner.
- This is because the merging order can cause two similar adjacent regions R_1 and R_2 not to be merged, since an earlier merge used R_1 and its new characteristics no longer allow it to be merged with regionR_2.
- If merging process used a different order, this merge may have been realized.
- The simplest methods begin merging by starting the segmentation using regions of 2x2, 4x4, or 8x8 pixels.
- Region description is based on gray level properties.
- A region description is compared with the adjacent region; if they match, they are merged into larger region otherwise regions are marked as non-matching.

- Merging of adjacent regions continues between all neighbors, including newly formed ones. If a region cannot be merged is marked as 'final', when all image region are marked so, merging is stopped.

Algorithm : Region merging via boundary melting

1. Define starting image segmentation into regions of constant gray-level. Construct a super grid edge data structure in which to store the crack edge information.
2. Remove all weak crack edges from the edge data structure.
3. Recursively remove common boundaries of adjacent regions R_i, R_j , if

$$\frac{W}{\min(l_i, l_j)} \geq T_2$$

where w is the number of weak edges on the common boundary and l_i, l_j are the perimeter lengths of regions R_i, R_j and T_2 is another preset threshold.

4. Recursively remove common boundaries of adjacent regions R_i, R_j if

$$\frac{W}{l} \geq T_3$$

Or, using a weaker criterion $W \geq T_3$ where l is the length of the common boundary and T_3 is a third threshold.

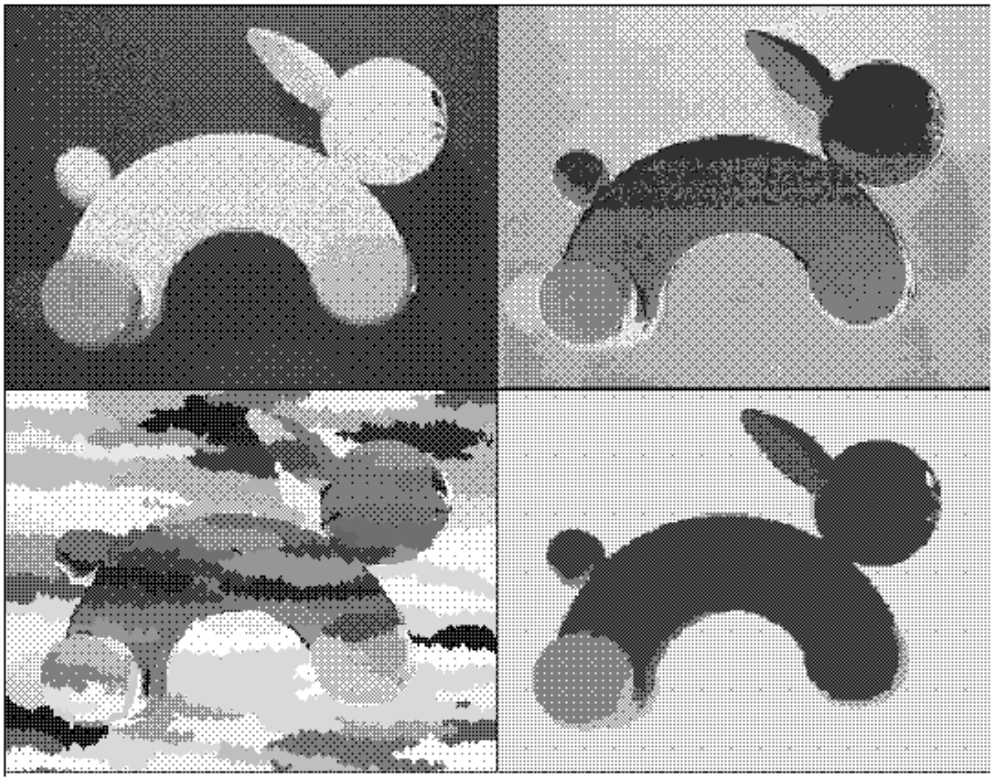


Fig 2: Region merging segmentation (a)Original image (b)Pseudo-color representation of the original image (c)Recursive region merging (d)Region merging via boundary melting.

Region Splitting

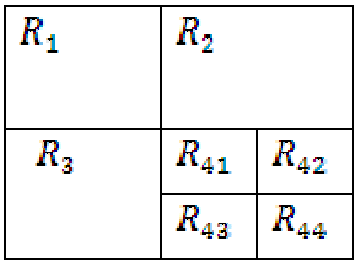
- Region splitting is the opposite of region merging, and begins with the whole image represented as a single image. Region splitting operations add missing boundaries by splitting regions that contain parts of different objects.
- Splitting schemes begin with a partition, example the whole image

  $P(R_i \cup R_j)$=FALSE
- Then, they proceed to satisfy condition by gradually splitting image regions.

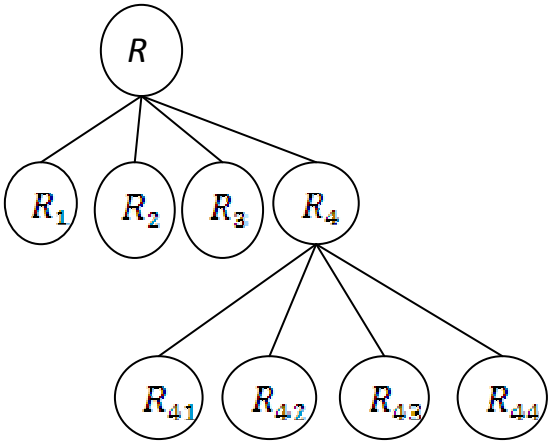
  $P(R_i)$= TRUE
- Two main difficulties in implementing this approach:
  - Deciding when to split a region (e.g., use variance, surface fitting).
  - Deciding how to split a region.

(1) If $P(R_i)$ =False, split *R* into four quadrants

(2) If *P* is False on any quadrant, sub split



(a)Partitioned image



(b)Corresponding quad tree

Region Splitting and Merging

- If only splitting were used, the final partition likely would contain adjacent regions with identical properties. This drawback may be remedied by allowing merging, as well as splitting.
- Split-and-merge approaches work using pyramid image representations; regions are square-shaped and correspond to elements of the appropriate pyramid level.

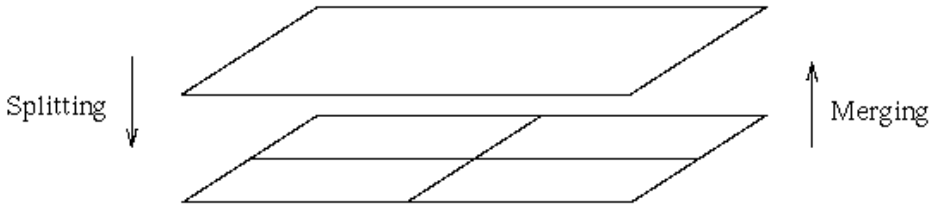


Fig 3: Split and merge in hierarchical data structure

- If any region in any pyramid level is not homogeneous (excluding the lowest level), it is split into four sub regions -- these are elements of higher resolution at the level below.
- If four regions exist at any pyramid level with approximately the same value of homogeneity measure, they are merged into a single region in an upper pyramid level.
- The segmentation process can be understood as the construction of a segmentation quad tree where each leaf node represents a homogeneous region.
- Splitting and merging corresponds to removing or building parts of the segmentation quad tree.
- Split-and-merge methods usually store the adjacency information in region adjacency graphs (or similar data structures).
- Using segmentation trees, in which regions do not have to be contiguous, is both implementationally and computationally easier.
- An unpleasant drawback of segmentation quad trees is the square region shape assumption

    - merging of regions which are not part of the same branch of the segmentation tree

- Because both split-and-merge processing options are available, the starting segmentation does not have to satisfy any of the homogeneity conditions.

Algorithm : Split and Merge

1. Define an initial segmentation into regions, a homogeneity criterion, and a pyramid data structure.
2. If any region R in the pyramid data structure is not homogeneous $[P(R) = FALSE]$, split it into four child-regions; if any four regions with same parent can be merged into a single homogeneous region, merge them. If no region can be split or merged, go to step 3.
3. If any two adjacent regions R_i, R_j can be merged into a homogeneous region, merge them.
4. Merge small regions with the most similar adjacent region if it is necessary to remove small-size regions.

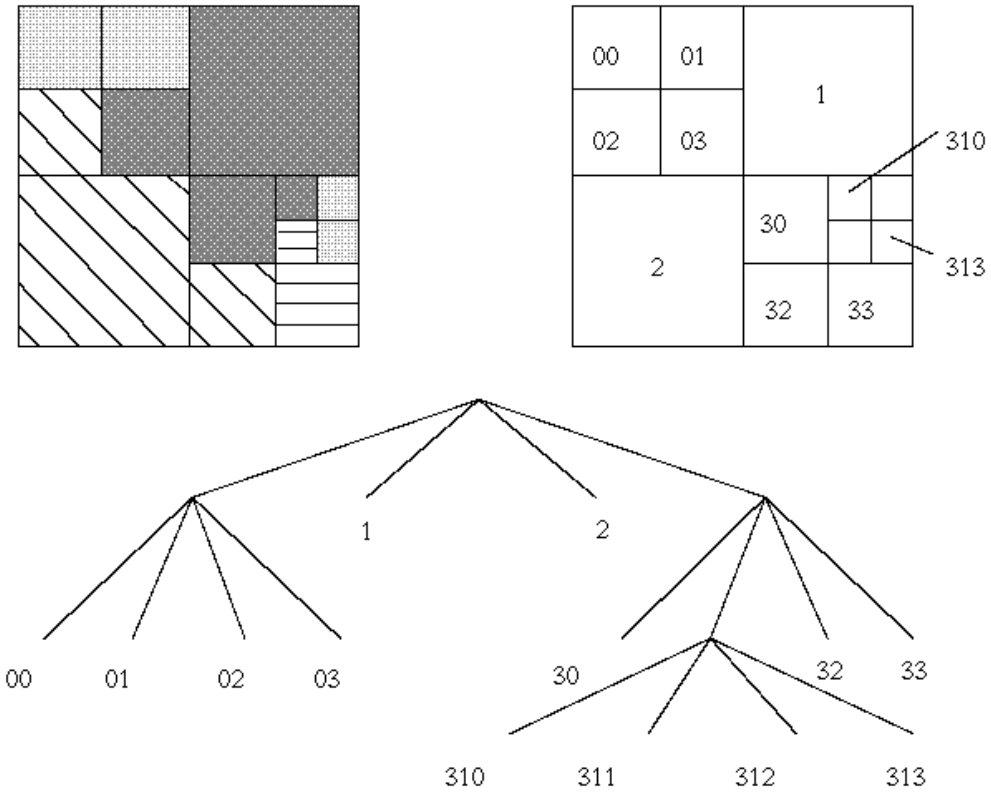


Fig 4: Segmentation quad tree

- A pyramid data structure with overlapping regions is an interesting modification of this method.
- Each region has four potential parent elements in the upper pyramid level and sixteen possible child elements in the lower pyramid level.
- Segmentation tree generation begins in the lowest pyramid level. Properties of each region are compared with properties of each of its potential parents and the segmentation branch is linked to the most similar of them.
- After construction of the tree is complete, all the homogeneity values of all the elements in the pyramid data structure are recomputed to be based on child-region properties only.
- This recomputed pyramid data structure is used to generate a new segmentation tree, beginning again at the lowest level.

- The pyramid updating process and new segmentation tree generation is repeated until no significant segmentation changes can be detected between steps.
- The highest level of the segmentation tree must correspond to the expected number of image regions and the pyramid height defines the maximum number of segmentation branches.
- If the number of regions in an image is less than 2^n, some regions can be represented by more than one element in the highest pyramid level.

Algorithm: Single-pass split and merge

1. Search an entire image line by line except the last column and last line. Perform the following steps for each pixel.
2. Find a splitting pattern for a 2x2 pixel block.
3. If a mismatch between assigned labels and splitting patterns in overlapping blocks is found, try to change the assigned labels of these blocks to remove the mismatch.
4. Assign labels to unassigned pixels to match a splitting pattern of the block.
5. Remove small regions if necessary.

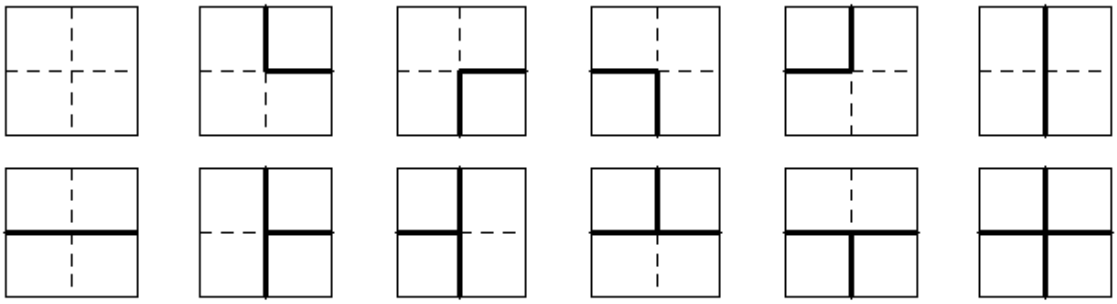


Fig 5 : Splitting of 2x2 image blocks, all 12 possible cases.

- The image blocks overlap during the image search.
- Except for locations at the image borders, three of the four pixels have been assigned a label in previous search locations, but these labels do not necessarily match the splitting pattern found in the processed block.

- If a mismatch is detected in step 3 of the algorithm, it is necessary to resolve possibilities of merging regions that were considered separate so far - to assign the same label to two regions previously labeled differently.
- Two regions R_1 and R_2 are merged into a region R_3 if

$$P(R_1 \cup R_2) = TRUE$$

$$|m_1 - m_2| < T\ ,$$

Where m_1 and m_2 are the mean gray level values in regions R_1 and R_2 , and T is some appropriate threshold.

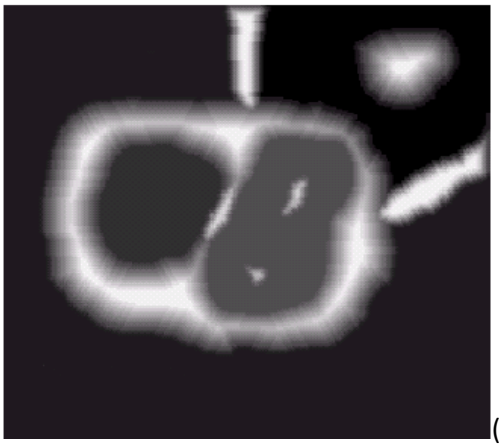
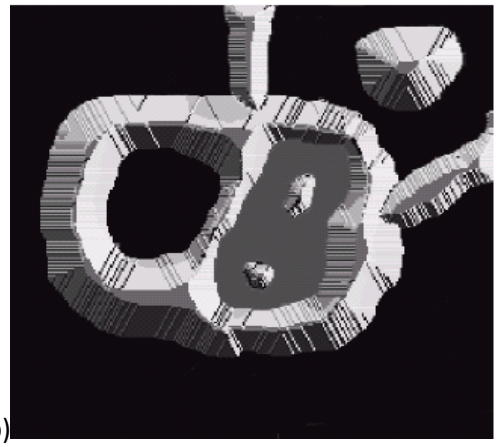
- If region merging is not allowed, regions keep their previous labels.
- If larger blocks are used, more complex image properties can be included in the homogeneity criteria (even if these larger blocks are divided into 2x2 sub-blocks to determine the splitting pattern).

Watersheds segmentation

- Watersheds are based on visualizing an image in three dimensions: two spatial coordinates versus gray levels. In such a topographic interpretation, we consider three types of points
  (a) Points belonging to a regional minimum,
  (b) Points at which a drop of water, if placed at the location of any of those points, would fall with certainty to a single minimum, (catchment basin or watershed of that minimum)
  (c) Points at which water would be equally likely to fall to more than one such minimum (divide lines or watershed lines).

- Segmentation algorithms based on these concepts is to find the watershed lines.

[Suppose that a hole is punched in each regional minimum and that the entire topography is flooded from below by letting water rise through the holes at a uniform rate. When the rising water in distinct catchment basins is about to merge, a dam is built to prevent the merging. The flooding will eventually reach a stage when only the tops of the dams are visible above the watersheds. Therefore, they are the boundaries extracted by a watershed segmentation algorithm.]

(a)  (b) 

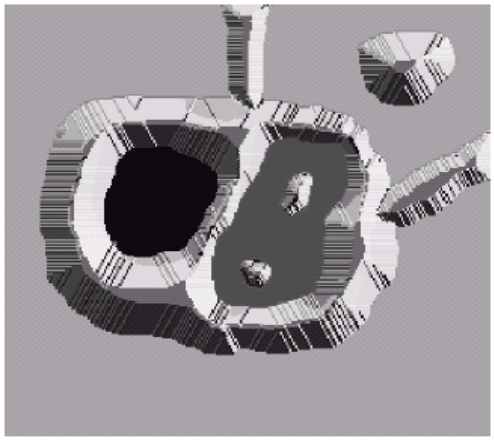
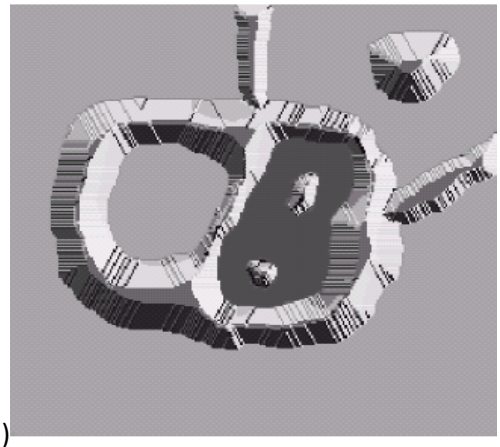
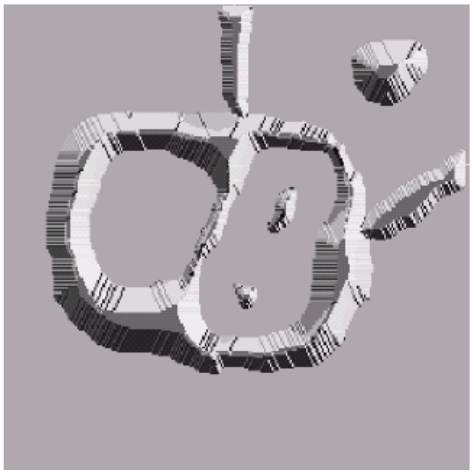
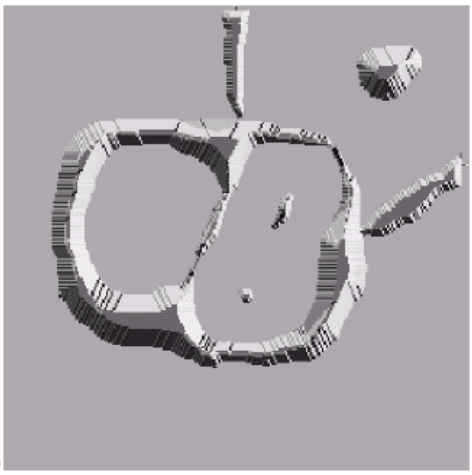
(c)  (d) 

Fig 1 [(a)Original image ,(b)Topographic view,(c)-(d) two stages of flooding].

(e)  (f) 

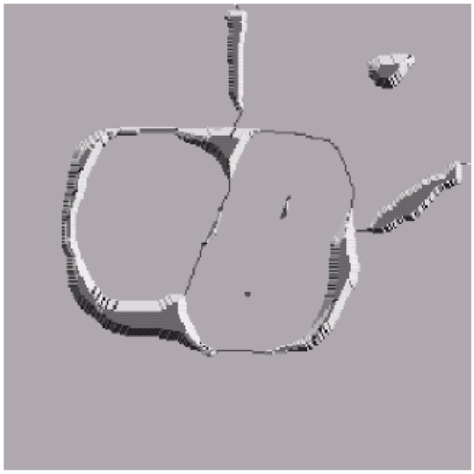
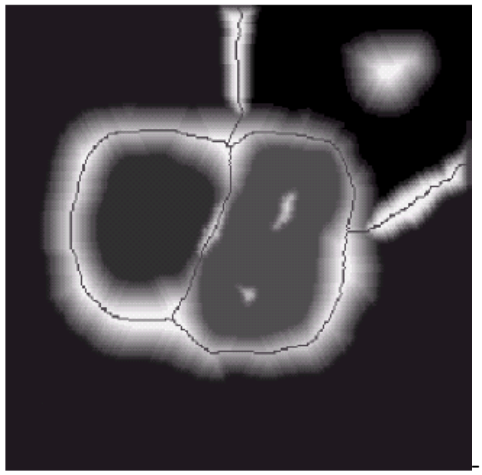
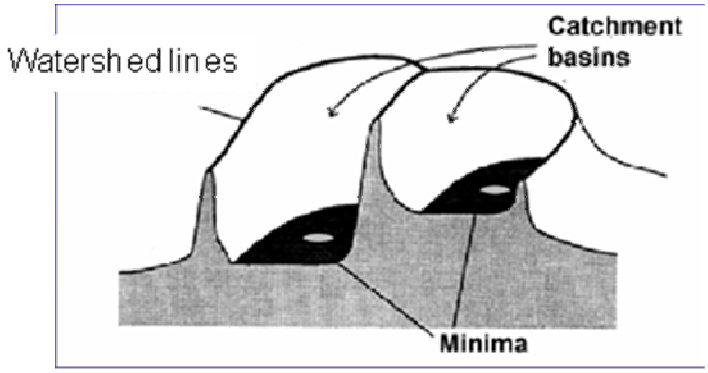
(g)  (h) 

Fig 1(e) Results further flooding (f) Beginning of merging of water from two catchment basins (g) longer dams (h) Final watershed.

- Fig 1(a) shows a simple gray-scale image and Fig (b) is a topographic view, in which the height of the mountains is proportional to gray-level values in the input image.
- In order to prevent the rising water from spilling out through the edges of the structure, we imagine the perimeter of entire topography being enclosed by dams of height greater than the highest possible mountain, whose value is determined by the highest possible gray-level value in the input image.

- Suppose that a hole is punched in each regional minimum (dark area in Fig 1(b)) and that the entire topography is flooded from below by letting water rise through the holes at a uniform rate.
- Fig1(c) shows the first stage of flooding where water shown in light gray.
- In Fig1 (d) and (e) water has risen into the first and second catchment basins. As the water continues to rise, it will overflow from one catchment basin to another as in Fig1 (f).
- Here dam was built to prevent water from merging at that level of flooding. Fig 1(g) shows a longer dam between the two catchment basins and another dam in the top part of the right basin.
- The later dam was built to prevent merging of water from that basin with water from areas corresponding to the background.
- This process is continued until the maximum level of flooding is reached. The final dams correspond to the watershed lines, which are the desired segmentation result is as shown in Fig1 (h).

[Note: Watershed lines form a connected path for continuous boundaries between regions].



- Watershed segmentation is used in the extraction of uniform objects from the background.
- Regions characterized by small variations in gray levels have small gradient values.
- Thus watershed segmentation is applied to the gradient of an image. In this the regional minima of catchment basins correlate with the small value of the gradient corresponding to the objects of interest.

Dam Construction

- Dam construction is based on binary images. The simplest way to construct dams separating sets of binary points is to use morphological dilation.

Basics to construct dam using dilation

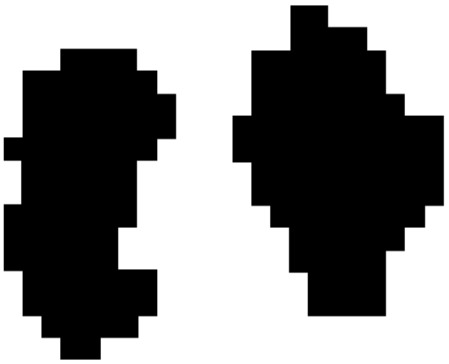


Fig 2(a) Two partially flooded
Catchment basins at stage *n-1* of flooding

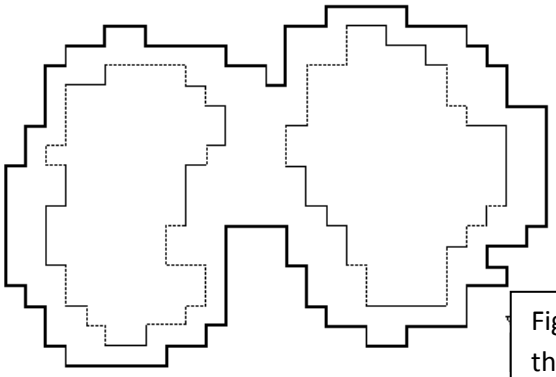


Fig 2(b) Flooding at stage *n*, showing that water has spilled between basins (for clarity, water is shown in white).

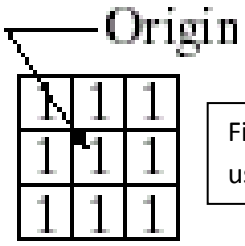


Fig 2(c) Structuring element used for dilation.

The basics to construct dams

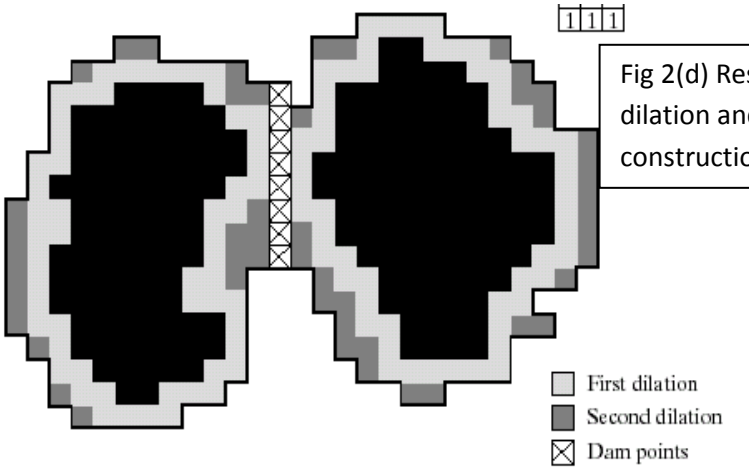


Fig 2(d) Result of dilation and dam construction

using

The basics to construct dams using dilation are illustrated in Fig 2.

- Fig 2(a) shows the portions of two catchment basins at flooding step *n-1* and Fig 2(b) shows the result at the flooding step, *n*.
- Dam should be constructed to avoid the spilling of water from one basin to the other.
- Let M_1 and M_2 denote the sets of coordinates of points in two regional minima. $C_{n-1}(M_1)$ and $C_{n-1}(M_2)$ denote the set of coordinates of points in the *catchment basin* associated with these two minima at stage *n-1* of flooding. These are the two black regions shown in Fig 2(a).
- *C[n-1]* be the union of these two sets. There are two connected components in Fig 2(a) and one connected component in Fig 2(b).
- Here two connected component encompasses to become single component indicates that water between the two catchment basins has merged at flooding step *n*. This is denoted by *q.*
- The two components from step *n-1* can be extracted from *q* by performing the simple *AND* operation $q \cap C[n-1]$. All points belonging to an individual catchment basin form a single connected component.
- The connected components is dilated as shown in Fig 2(c) a structuring element by two conditions,

1. The dilation has to be constrained to q (the center of the structuring element can be located only on points in q during dilation) .

2. The dilation cannot be performed on points that would cause the sets being dilated to merge (single component).

- Fig 2(d) shows that a first dilation pass (light gray) expanded the boundary of each original connected component. Condition 1 was satisfied by every point during dilation, and condition 2 did not apply to any point during the dilation process; thus the boundary of each region was expanded uniformly.
- In second dilation (medium gray), several points failed condition 1 while meeting condition 2, resulting in the broken perimeter shown in Figure. Points in q that satisfy the two conditions describes the one-pixel-thick connected path shown crossed-hatched in Fig 2(d) which constitutes the desired separating dam at stage n of flooding.
- Construction of dam at this level of flooding is completed by setting all the points in the path just determined to a value greater than the maximum gray-level value of an image.
- The height of all dams is generally set at 1 plus the maximum allowed value in the image.
- This will prevent water from crossing over the part of the completed dam as the level of flooding is increased.
- Dams build by this procedure are connected components (eliminates the problem of broken segmentation lines).

Watershed Segmentation Algorithm

1. Let $M_1, M_2, \ldots\ldots M_R$ be sets denoting the coordinates of the points in the regional minima of an image *g(x,y).*
2. Let $C(M_i)$ be a set denoting the coordinates of the points in the catchment basins associated with regional minimum M_R [min and max is used to

denote minimum and maximum]. $T[n]$ represents the set of coordinates (s,t) for which $g(s,t) < n$ i.e..

$$T[n] = \{(s,t) | g(s,t) < n\}$$

$T[n]$ is the set of coordinates of points in $g(x,y)$ lying below the plane $g(x,y) = n$.

3. The topography will be flooded in integer flood increments, from n =min+1 to n=max+1.In flooding process, the algorithm needs to know the number of points below the flood depth. Suppose the coordinates in $T[n]$ that are below the plane $g(x,y) = n$ are marked black, and all other coordinates are marked white.
4. Let $C_n(M_i)$ denote the set of coordinates of points in the catchment basin associated with minimum M_i that are flooded at stage n. Binary image of $C_n(M_i)$

$$C_n(M_i) = C(M_i) \cap T[n].$$

$C_n(M_i)$ =1 at location (x,y) if $(x,y) \in C(M_i)$ AND $(x,y) \in T[n]$ otherwise $C_n(M_i) = 0$.

$C[n]$ denote the union of the flooded catchment basins portions at stage n :

$$C[n] = \cup_{i=1}^{R} C_n(M_i)$$

$C[max+1]$ is the union of all catchment basins:

$$C[max+1] = \cup_{i=1}^{R} C_n(M_i)$$

Elements in both $C_n(M_i)$ and $T[n]$ are never replaced during execution of algorithm and that two elements in these two sets either increases or remains the same as n increases.

- $C[n-1]$ is a subset $C[n]$

- $C[n]$ is subset of $T[n]$
- $C[n-1]$ is subset of $T[n]$

  From this the result is each connected component of $C[n-1]$ is contained exactly one connected component of $T[n]$.

5. The algorithm for finding the watershed lines is initialized with

$C[min+1]= T[min+1]$. The algorithm then proceeds recursively, assuming at step n that $C[n-1]$ has been constructed. A procedure for obtaining $C[n]$ from $C[n-1]$ is as follows. Q denote the set of connected components in $T[n]$.

6. For each connected component $\in Q[n]$, there are three possibilities:

(a) $q \cap C[n-1]$ is empty.

(b) $q \cap C[n-1]$ contains one connected component of $C[n-1]$.

(c) $q \cap C[n-1]$ contains more than one connected component of $C[n-1]$.

Construction of $C[n]$ from $C[n-1]$ depends on which of these three conditions holds. Condition (a) occurs when a new minimum is encountered, in which case connected component *q* is incorporated into $C[n-1]$ to form $C[n]$. Condition (b) occurs when *q* lies within the catchment basin of some regional minimum, in which case *q* is incorporated into $C[n-1]$ to form $C[n]$. Condition (c) occurs when all, or part, of a ridge separating two or more catchment basins is encountered.

- Further flooding would cause the water level in these catchment basins to merge. Thus a dam must be built within *q* to prevent overflow between the catchment basins.
- Algorithm efficiency is improved by using only values of *n* that correspond to existing gray-level values in $g(x,y)$; we can determine these values, as well as the values of min and max, from the histogram of $g(x,y)$.

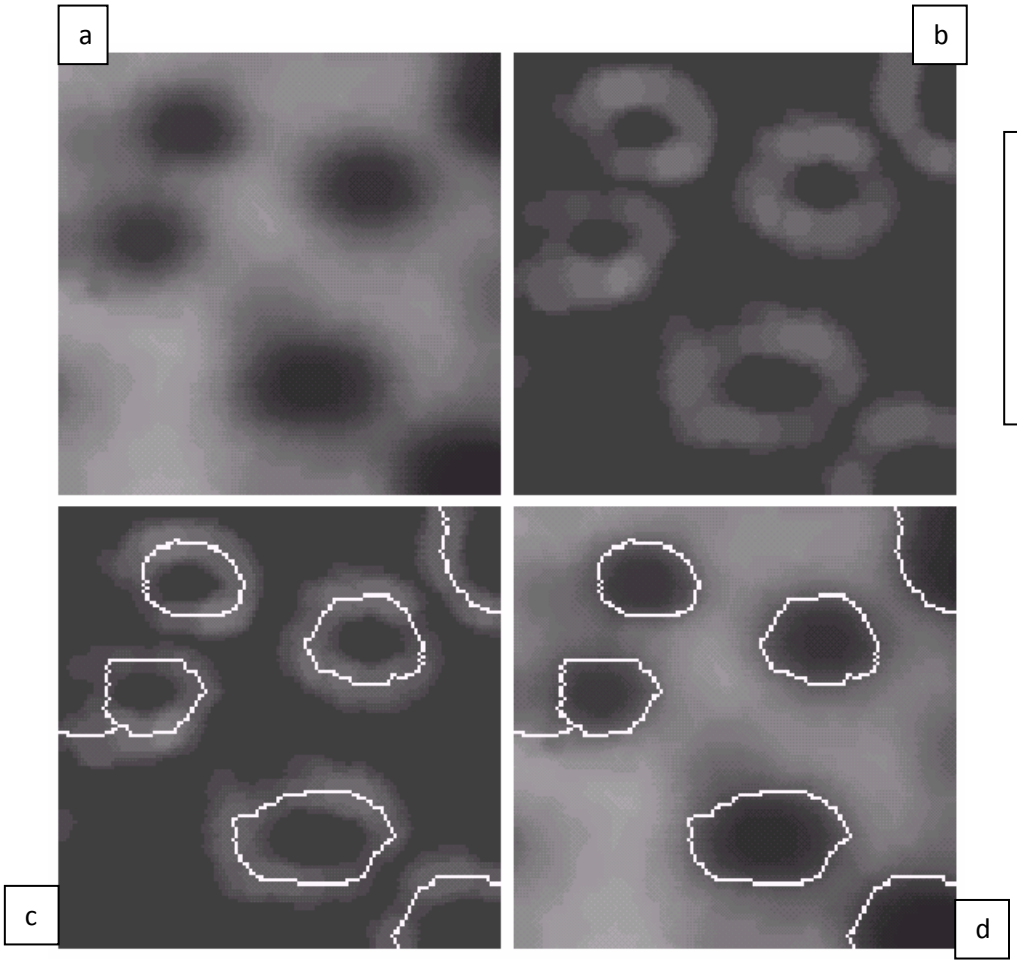


Fig 3 (a) Image of blobs
(b) Image Gradient
(c)Watershed lines
(d)Watershed lines superimposed on original image

The Use of Markers

- Problem with basic watershed algorithm is *over segmentation* due to noise and other local irregularities of the gradient.
- This results in making a large number of segmented regions useless.
- Solution to this problem is to limit the number of allowable regions by incorporating a preprocessing stage designed to bring additional knowledge into the segmentation procedure.
- Over segmentation can be control by a concept called markers. A *marker* is a connected component belonging to an image.
- *Internal* marker is used for objects and *external* marker for background.
- Two principal steps of marker selection

    1. Preprocessing

2. Definition of a set of criteria that markers must satisfy.

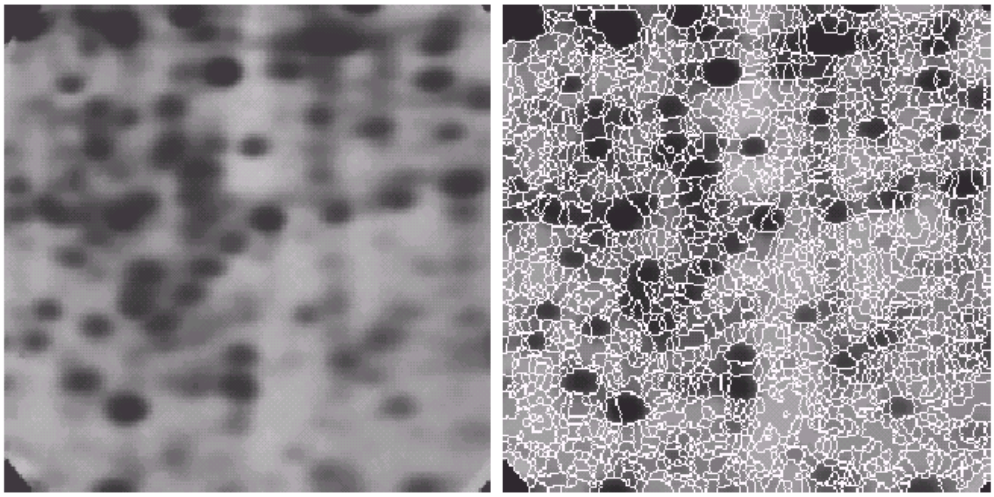


Fig 4 (a) Electrophoresis image, (b) Result of applying watershed segmentation algorithm to the gradient image (over segmentation).

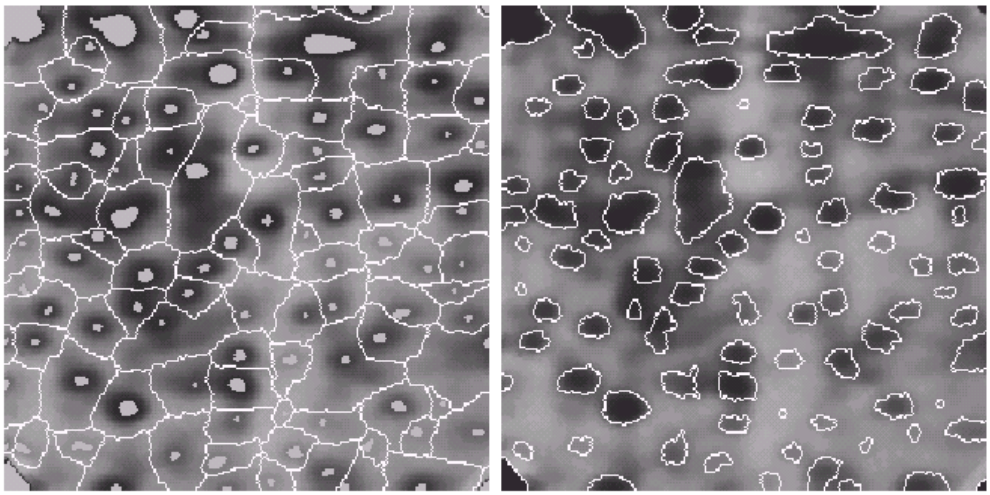


Fig 5 (a) internal marker image (light gray regions) and external markers (watershed lines) (b) Result of segmentation.

- We define an internal marker (1) a region that is surrounded by points of higher altitude (2) such that the points in the region form a connected component and (3) in which all the points in the connected component have the same gray-level value.
- After the image was smoothed, the internal markers resulting from this definition are shown as light gray, blob like region in Fig 5(a).

- Next watershed algorithm was applied to the smoothed image, the restriction that these internal markers be the only allowed regional minima.
- Fig 5(a) shows the resulting watershed lines defined as the external markers. The external markers partition the image into regions, with each region containing a single internal marker and part of the background.
- Now apply watershed segmentation algorithm to each individual region; Fig 5 (b).
- The point is that using markers brings a priori knowledge to bear on the segmentation problem.
- The fact that segmentation by watersheds offers a framework that can make effective use of this type of knowledge is a significant advantage of this method.

Matching

- Matching is another basic approach to segmentation that can be used to locate known objects in an image, to search for specific patterns, etc.
- The best match is based on some criterion of optimality which depends on object properties and object relations.

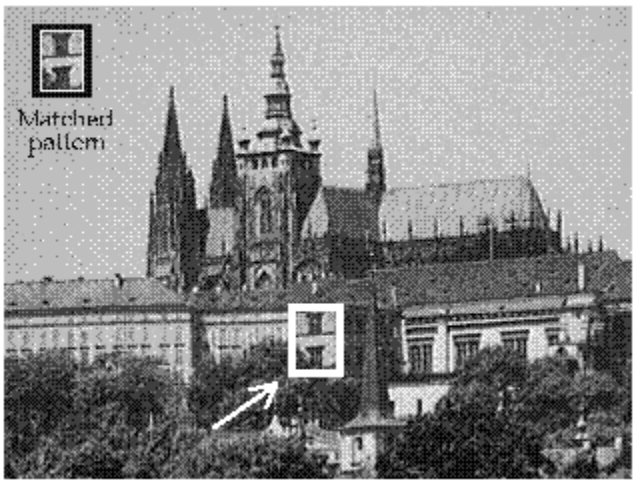


Fig 1: Segmentation by matching; matched pattern and location of the best match.

- Matched patterns can be very small, or they can represent whole objects of interest.

- While matching is often based on directly comparing gray-level properties of image sub regions, it can be equally well performed using image-derived features or higher-level image descriptors.

- In such cases, the matching may become invariant to image transforms.

- Criteria of optimality can compute anything from simple correlations up to complex approaches of graph matching.

Matching criteria

- Exact copy of the pattern of interest cannot be expected in the processed image - some part of the pattern is usually corrupted in real images by noise, geometric distortion, occlusion, etc.
- Search for locations of maximum match is appropriate.

Match based segmentation Algorithm

1. Evaluate a match criterion for each location and rotation of the pattern in the image.
2. Local maxima of this criterion exceeding a present threshold represent pattern locations in the image.

- Matching criteria can be defined in many ways; in particular, correlation between a pattern and the searched image data is a general matching criterion.

- Let f be an image to process, h be a pattern to search for, and V be the set of all image pixels in h.
- Possible matching optimality criteria describing a match between f and h located at a position (u,v):

$$C_1(u,v) = \frac{1}{1+max_{(i,j)\in V}\ |f(i+u,j+v)-h(i,j)|},$$

$$C_2(u,v) = \frac{1}{1+\sum_{(i,j)\in V}|f(i+u,j+v)-h(i,j)|},$$

$$C_3(u,v) = \frac{1}{1+\sum_{(i,j)\in V}(f(i+u,j+v)-h(i,j))^2}.$$

- The "1" added to each denominator to prevents dividing by zero for a perfect match.
- The cost is evaluated at each (u,v) pixel location in the image being processed.
- Possible implementation decisions include whether the pattern is only computed entirely within the image or partial pattern positions when the pattern crosses image borders.

- A simple example of the C_3 optimality criterion values is given:

1	1	0	0	0
1	1	1	0	0
1	0	1	0	0
0	0	0	0	0
0	0	0	0	8

(a)

1	1	1
1	1	1
1	1	1

(b)

1/3	1/6	1/8	×	×
1/5	1/7	1/8	×	×
1/8	1/9	1/57	×	×
×	×	×	×	×
×	×	×	×	×

(c)

Fig 2: Optimality matching criterion evaluation (a)Image data (b)Matched pattern (c)Values of the optimality criterion C_3

- If a fast, effective Fourier transform algorithm is available, the convolution theorem can be used to evaluate matching.
- The correlation between a pattern h and image f can be determined by first taking the product of the Fourier transform F of the image f and the complex conjugate of the Fourier transform $H^{\#}$of the pattern h and then applying the inverse transform.

- To compute the product of Fourier transforms, F and $H^{\#}$ must be of the same size; if a pattern size is smaller, zero-valued lines and columns can be added to inflate it to the appropriate size.
- Sometimes, it may be better to add non-zero numbers, for example, the average gray level of processed images can serve the purpose well.

Control strategies of matching

- Match-based segmentation localizes all image positions at which close copies of the searched pattern are located.
- These copies must match the pattern in size and rotation, and the geometric distortion must be small.
- To adapt match-based methods to detect patterns that are rotated, enlarged, and/or reduced, it would be necessary to consider patterns of all possible sizes and rotations.
- Another option is to use just one pattern and match an image with all possible geometric transforms of this pattern, and this may work well if some information about the probable geometric distortion is available.
- Note that there is no difference in principle between these approaches.

- Matching can be used even if an infinite number of transformations are allowed. Let us suppose a pattern consists of parts, these parts being connected by rubber links.
- Even if a complete match of the whole pattern within an image may be impossible, good matches can often be found between pattern parts and image parts.
- Good matching locations may not be found in the correct relative positions, and to achieve a better match, the rubber connections between pattern parts must be either pushed or pulled.

- The final goal can be described as the search for good partial matches of pattern parts in locations that cause minimum force in rubber link connections between these parts.
- A good strategy is to look for the best partial matches first, followed by a heuristic graph construction of the best combination of these partial matches in which graph nodes represent pattern parts.

- Match-based segmentation is time consuming even in the simplest cases with no geometric transformations, but the process can be made faster if a good operation sequence is found.
- The sequence of match tests must be data driven.
    - Fast testing of image locations with a high probability of match may be the first step, then it is not necessary to test all possible pattern locations.
    - Another speed improvement can be realized if a mismatch can be detected before all the corresponding pixels have been tested.
    - The correlation changes slowly around the best matching location ... matching can be tested at lower resolution first, looking for an exact match in the neighborhood of good low-resolution matches only.

- The mismatch must be detected as soon as possible since mismatches are found much more often than matches.
- Considering the matching formulae given above, testing in a specified position must stop when the value in the denominator (measure of mismatch) exceeds some preset threshold.

- This implies that it is better to begin the correlation test in pixels with a high probability of mismatch in order to get a steep growth in the mismatch criterion.
- This criterion growth will be faster than that produced by an arbitrary pixel order computation.