

Segmentation

Segmentation is the first step in analysis or automatic interpretation of an image. It partitions an image into distinct regions that contain objects or features of interest. Segmentation can be regarded as the process of grouping together pixels that have similar attributes.

Segmentation is the intermediate stage between low level and high level image processing. The former manipulates pixel values to correct defects or enhance the image, whereas the latter manipulates and analyses a group of pixels that represent a particular feature of interest. Segmentation is used in a variety of applications such as

- Industrial inspection
- Optical character recognition
- Tracking of objects in a sequence of images
- Classification of terrain in remote sensing situations
- Detection and measurement of bone, tissue, etc. in medical images

Segmentation techniques can be categorized as either based purely pixel values, or a combination of pixel values and their locations.

6.1 Thresholding

This technique belongs to the first category, and is used in a variety of image processing operations. Thresholding is a segmentation technique because it classifies pixels into two groups – those at which some property (e.g. gray level) falls below a threshold, and those at which the property equals or exceeds the threshold. Because there are two outcomes, the resulting output image will be a binary image. Note that we have already used thresholding following edge detection to classify the pixels as edge pixels or background pixels (Section 4.2.3).

In thresholding, the output image $g(x,y)$ is obtained from the input image as

$$g(x,y) = \begin{cases} Lo, & f(x,y) < T \\ Hi, & f(x,y) > T \end{cases} \quad (6.1)$$

where Lo is a low gray level value (say 0), and Hi is a high gray level value (say 255), and T is the threshold value. Note that thresholding can be performed in-place. Fig. 6.1 shows the image of a Mars terrain, with a threshold of $T = 170$, $Lo = 0$, and $Hi = 255$ applied to detect rocks from the background.

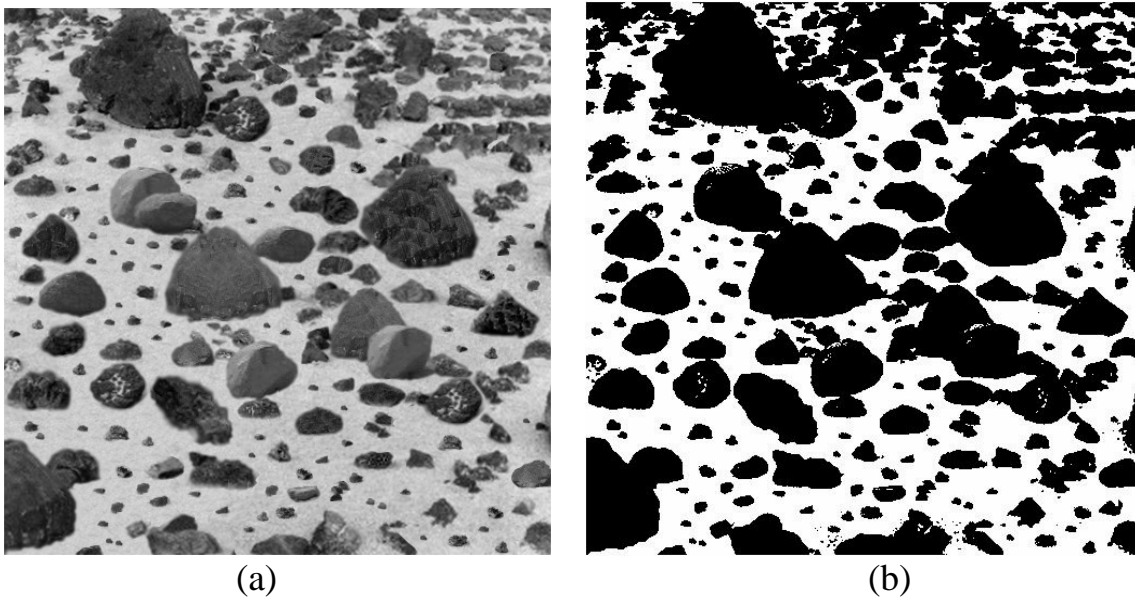


Fig. 6.1 Applying thresholding to a Mars terrain image to detect rocks.

We can use two thresholds to define an acceptable range for the object to be detected. This is written as

$$g(x,y) = \begin{cases} Lo, & f(x,y) < T_1 \\ Hi & T_1 \leq f(x,y) \leq T_2 \\ Lo & f(x,y) > T_2 \end{cases} \quad (6.2)$$

The success of thresholding depends critically on the selection of the threshold. Consider the regions in Fig. 6.2a where regions have varying gray levels. Choosing a threshold of $T = 128$ gives the image of Fig. 6.2b.

On the other hand with $T = 225$, only one region is identified and the remaining the regions are discarded, e.g. set to background (Fig. 6.2c).

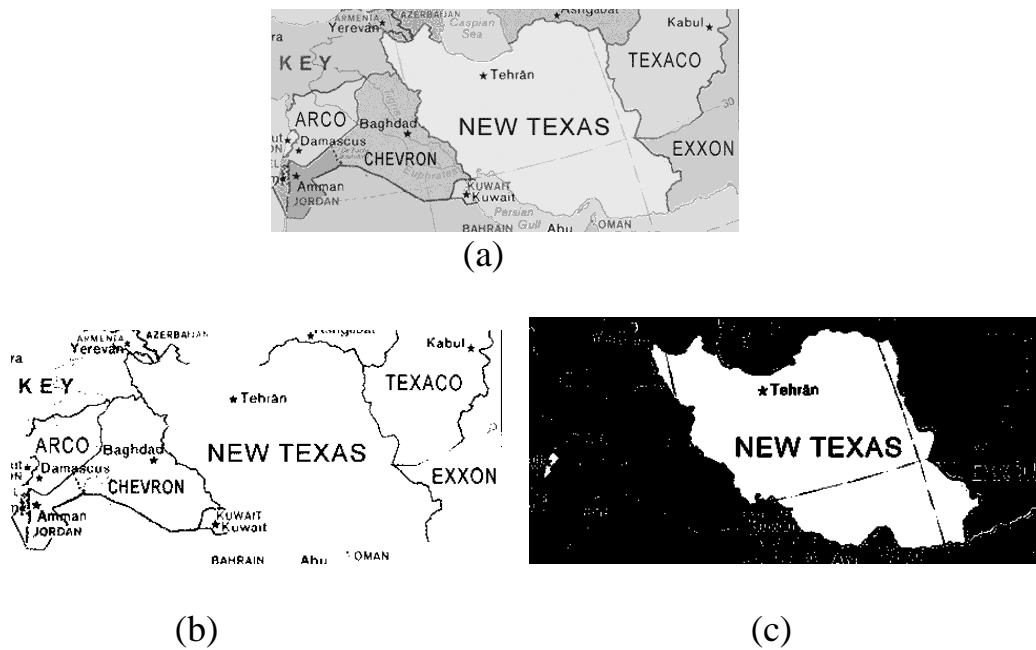


Fig. 6.2 The effects of various thresholding values

For automatic determination of the threshold, one simple approach is to make T equal to the mean gray level of the image. The idea is that mean lies between the pixel values that belong to the region (objects of interest) and the background (other things not of interest). This approach is effective for images that have bright objects on a simple, dark background, or vice versa.

A more sophisticated approach is to base the threshold value(s) on histogram analysis. If the histogram has distinct peaks and valleys, then we may distinguish between two features of different gray level by thresholding at a point between histogram peaks corresponding to these features. Fig. 6.3 shows a histogram with distinct peaks and valleys, and possible choices for the threshold values.

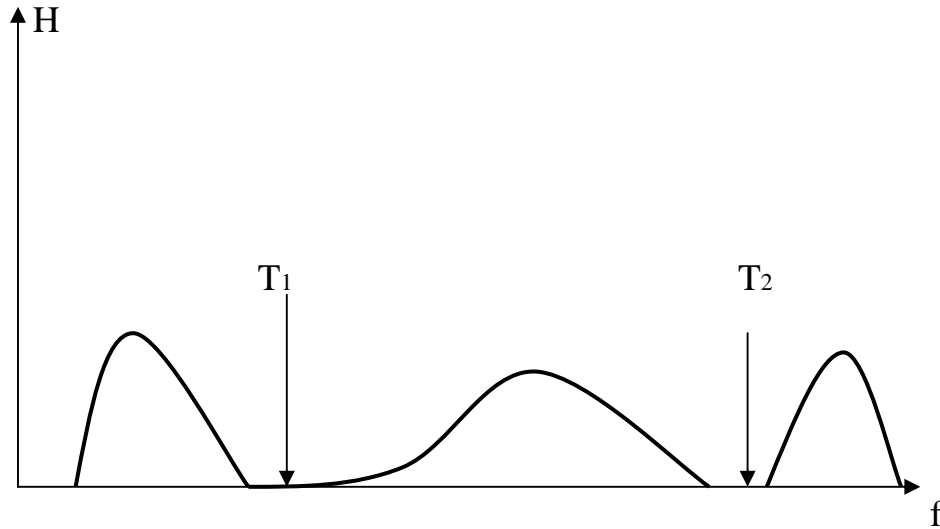


Fig. 6.3 Determining threshold values from the histogram.

The difficulty with the above method is that the histogram segments overlap, and peaks/valleys are in general not so distinct due to smoothly varying gray level or color values in an object. To partly remedy this situation, an iterative method for automatic threshold selection can be used, as in the following algorithm. The method starts with an initial guess of the threshold and improves this estimate by successive passes through the image. The initial value is usually found as the average of the gray level values of corner pixels which are assumed to be the background, and the average of all other pixels. Only few iterations are needed for the algorithm to converge to the final threshold.

Compute $m1$, the mean gray level of the corner pixels

Compute $m2$, the mean gray level of other pixels

$T_{old} = 0$

$T_{new} = (m1+m2)/2$

while ($T_{new} \neq T_{old}$) {

$m1$ = mean gray level of pixels for which $f(x,y) < T_{new}$

$m2$ = mean gray level of pixels for which $f(x,y) \geq T_{new}$

$T_{old} = T_{new}$

$T_{new} = (m1 + m2)/2$

}

6.1.1 Thresholding of Color Images

The simplest approach to color images is to define three thresholds, one for each color. A particular colors in an RGB image can be visualized as a point in the 3-D color space. A blue threshold, for example, can be visualized as a plane perpendicular to the B axis and parallel to the R-G plane. Thus the three thresholds divide the RGB space into 8 rectangular volumes, only one of which belongs to the object.

Fig. 6.4 shows thresholding where the red components in excess of 150 are set to white indicating the feature of interest while the other two components are set to 255 meaning that any blue or green component is set to zero.



Fig 6.4 The original image, and detected region using color thresholding

Another approach to thresholding is to define a spherical region in the RGB plane, instead of the above rectangular regions. The center of the sphere is specified by the point (R_0, G_0, B_0) , and the radius by ρ_0 . Any value within this sphere is set to a low value (say 0 black), and outside the sphere is set to a high value (say 255 white). This procedure is therefore formulated as

$$g(x, y) = \begin{cases} Lo, & \rho \leq \rho_0 \\ Hi, & \rho > \rho_0 \end{cases} \quad (6.3)$$

where $\rho = \sqrt{(f_R(x, y) - R_0)^2 + (f_G(x, y) - G_0)^2 + (f_B(x, y) - B_0)^2}$, and $f_R(x, y)$ is the red component of the pixel at location x,y, and similarly the other two color components are defined. Fig. 6.5 show the results of this thresholding on the fish image of Fig. 6.4a with $R_0 = G_0 = B_0 = 50$ and $\rho_0 = 170$.



Fig. 6.5 The results of distance thresholding on Fig. 6.4a

Note that we can generalize (6.3) by defining different thresholds on each RGB component. In this case the defined volume becomes an ellipsoid in the RGB space.

Finally, we can employ histogram for as an aid to determine the threshold for each of the RGB components. Since 3-D histograms are difficult to visualize, the 2-D projections, e.g. in R-G plane, can be used for this purpose. The procedure is then to determine peaks and valleys as before in these 2-D planes.

6.2 Segmentation Based Similarity or Discontinuity

Thresholding techniques discussed above, group pixels according to some global criterion such as gray level. Two pixels maybe far apart in an image and still be classified as belonging to the same region if their values satisfy the thresholding criterion. Another class of segmentation techniques is based on discontinuity or similarity of pixels in the image. Techniques based on discontinuity attempt to partition the image by detecting sudden changes. Edge detection fall into these techniques. On the other hand techniques based on similarity attempt to identify regions by grouping pixels that are “connected” and satisfy certain similarity criterion. We must first explain connectivity of pixels.

6.3 Pixel Connectivity

In general there are two types of neighborhood for a pixel. The 4-neighbors of the pixel at (x,y) consist of pixels located at $(x-1, y)$, $(x+1, y)$, $(x, y-1)$ and $(x, y+1)$, as shown in Fig. 6.6a. The 8-neighbors of the pixel at (x,y) are all surrounding pixels which are located at $(x+i, y+j)$, $i = -1,0,+1$; $j = -1,0,+1$, as shown in Fig. 6.6b.

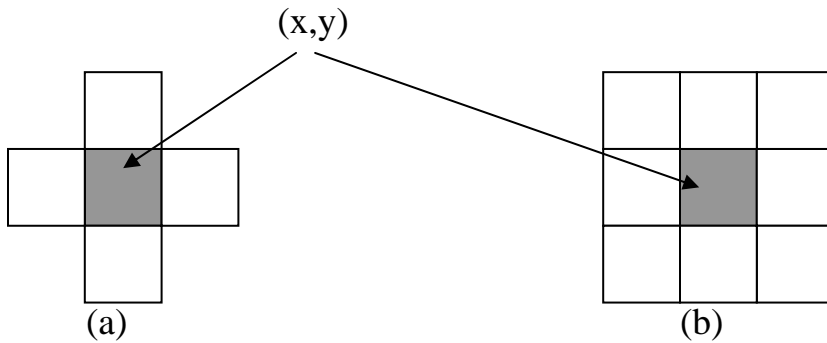


Fig. 6.6 The 4- and 8-neighbors of a pixel

A 4-connected path from a pixel p_1 to another pixel p_n is the sequence of pixels p_1, p_2, \dots, p_n such that p_{i+1} is a 4-neighbor of p_i for $i = 1, 2, \dots, n - 1$. This path is called an 8-connected path if p_{i+1} is an 8-neighbor of p_i .

Now in order to find a region, we apply thresholding and then pick 4-connected (or 8-connected) path from the thresholded pixels. As an example, consider the following image where the light and dark pixels are those that have gray levels more than a threshold. However, only dark pixels are selected for forming a region if 4-neighbor path is to be considered.

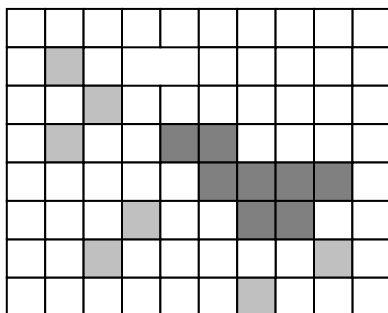


Fig. 6.7 The set of thresholded and 4-connected pixels.

The following Java code identifies and labels regions in an image.

```

public BufferedImage filter (BufferedImage input, BufferedImage output) {

    w = input.getWidth(); //assign dimension
    h = input.getheight(); // assign dimension
    WritableRaster in = input.copyData(null); //create a raster copy since
        // pixels are erased by “grassfire”
    WritableRaster out = output.getRaster();

    int n = 1; //initialize region number- 1st non-zero pixel and all
        // non-zero pixels connected to it is labeled 1
    for (int y=0; y < h; ++y) // now scan all pixels
        for(int x=0; x < w; ++x)
            if(in.getSample(x,y,0) >0 { //assume thresholded pixel>0. If
                // a non-zero pixel is found
                label(in, out, x, y, n); // invoke recursive labeling
                ++n; // increment the region number
                if( n > MAX_REGIONS) // terminate the labeling if more
                    return output; } // than MAX_REGIONS found
    return output; }

private void label (WritableRaster in, WritableRaster out, int x, int y, int n)
{
    in.setSample(x,y,0,0); // burn away pixels
    put.setSample(x,y,0,n); // set corresponding output pixel to region #
    int j,k;
    for (int i = 0; i < connectivity; ++i){ //examine pixel’s neighbors
        j = x + delta[i].x; // neighbors located at displacement delta.
        k = y + delat[i].y; // delta array has 4 or eight elements
        if( inImage(j,k) && in.getSample(j,k,0) > 0) // if neighbor coordinates
            // lie within image and the neighbor is non-zero
            label(in, out, j, k, n); } // label neighboring pixels
}

private final boolean inImage(int x, int y) {
    return x >= 0 && x < width && y >= 0 && y < height;
}

```

6.4 Region Similarity

The similarity of pixels in a region R is identified by a uniformity predicate $P(R)$ such as

$$P(R) = \begin{cases} \text{TRUE}, & \text{if } |f(i,j) - f(k,l)| \leq \Delta \\ \text{FALSE}, & \text{otherwise} \end{cases} \quad (6.4)$$

where $f(i,j)$ and $f(k,l)$ are the coordinates of pixels in region R . Note that that above does not mean that all pixel values in the region R are within Δ of each other. In fact, with (6.4) the difference in gray levels between pixel in a region can be much larger than Δ (why?).

A similar predicate is

$$P(R) = \begin{cases} \text{TRUE}, & \text{if } |f(i,j) - \mu_R| \leq \Delta \\ \text{FALSE}, & \text{otherwise} \end{cases} \quad (6.5)$$

where μ_R is the mean values of all pixels within the region. Equation (6.4)-(6.5) can be generalized to cope with color images where we now compute the distance in RGB space between the colors of neighboring pixels, or between the color of a pixel and mean color for the region.

6.4.1 Region Growing

Region growing starts by planting a set of seed pixels each of which is then grown to a uniform connected region. A pixel is added to a region if and only if

- has not been assign to any other region
- is a neighbor of that region
- the new region created by addition of the pixel is still uniform.

Region Growing Algorithm

```
Define a set of regions R1, R2,..., Rm each consisting a single seed pixel
while (no more pixels being assigned to region) {
  for(i = 1; i<m; i++)
    for(each pixel p at border of Ri)
```

```

for(all neighbors of p) {
  Let x,y be the neighbor coordinates
  Let mu_i be mean gray level of pixels in Ri
  if ( (neighbor is unassigned) && (abs (f(x,y) - mu_i) <= delta) {
    add neighbor to Ri
    update mu_i
  }
}

```

Fig. 6.8 shows an example of region growing using (6.5) with $\Delta = 3$, and 8-neighbor connectivity where the seeds are placed in pixels (2,1) and (2,3).

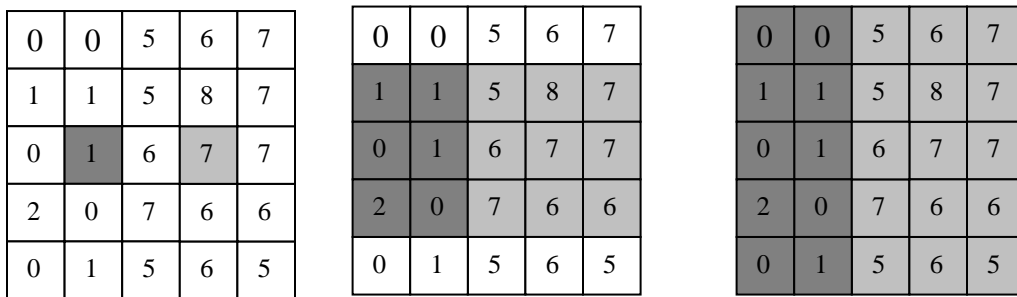
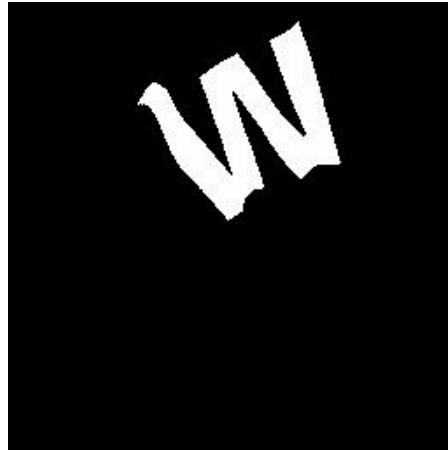


Fig. 6.7 Region growing

Region growing can be very effective in finding regions of interest. Fig. 6.8b is the result of region growing on the 256 by 256 image in Fig. 6.8a where one seed is place at the location (128, 100) using 4-connectivity and $\Delta = 30$. It is seen the region of interest, i.e. W, has been nicely segmented from the remaining of the image. This would not have been possible with thresholding (why?).

However, region growing suffers from a number of limitations. For example region growing with 4-connectivity can produce very different results from 8-connectivity. Also, the results obtained can be very sensitive to the choice of uniformity predicate, seed location, and Δ .



(a)

(b)

Fig. 6.8 Detecting a region with the region growing technique.

Fig 6.9a shows region growing on image of Fig. 6.8a with seed at (128,140), 4-connectivity and $\Delta = 30$. If we now use the same data on the seed location and 4-connectivity, but increasing Δ to 50, we get the result shown in Fig. 6.9b which shows the sensitivity of region growing to the tolerance Δ



(a)



(b)

Fig. 6.9 (a) and (b) showing the result of change in Δ

For a successful and complete segmentation, the following criteria must be satisfied.

- All pixels must be assigned to regions.
- Each pixel must belong to a single region only.
- Each region must be a connected set of pixels.
- Each region must be uniform.
- Any merged pair of adjacent regions must be non-uniform.

Region growing satisfies the third and fourth criteria but not the others. It fails to satisfy the first and second criteria because the number of seed is usually insufficient to create regions for every pixel. The fifth criterion is also not satisfied in the case of region growing (why?).

6.4.2 The Split and Merge

A complete segmentation is possible if we initially consider the entire image consider the entire image as a single region. Then we test this region for uniformity predicate, and if it is false, then the region is split or divided into sub-regions, each of which is tested for uniformity. The procedure iterates until all regions are uniform, or desired number of regions have been established.

A common strategy is to divide the image into 4 regions recursively, and check that for the region R_i , $P(R_i)$ is true. Each of these quadrants is in turn divided into 4 quadrants. At each stage the adjacent regions are checked, and if they have identical quality, then they are merged. Therefore, each split is followed by a test of adjacent regions and a possible merge. For example if R_i and R_j are two adjacent regions, then they are combined into a larger region if the uniformity predicate is true for the union of these two regions, i.e. $P(R_i \cup R_j) = \text{TRUE}$.

6.5 Region Representation

Once the regions are detected, they must be stored for possible further processing by the computer.

Occupancy Array: The simplest representation is the spatial occupancy array which defines a membership function as

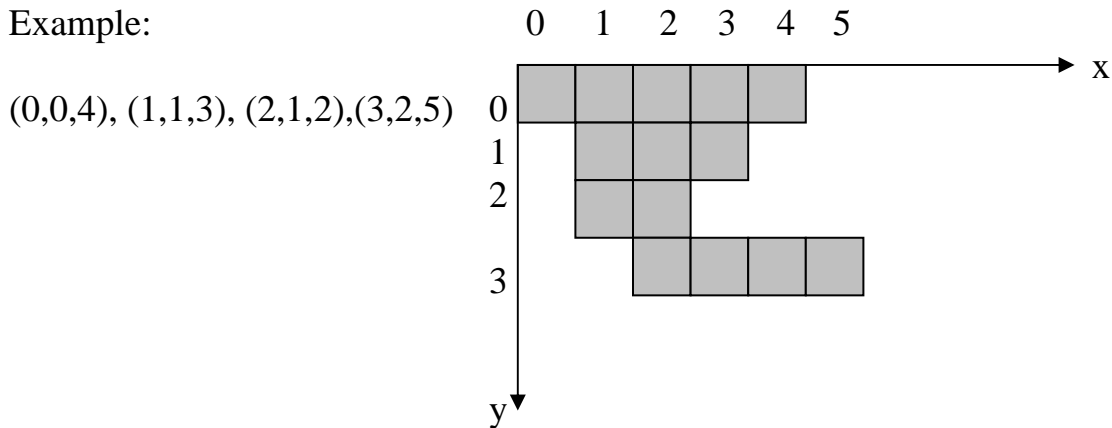
$$R(x, y) = \begin{cases} 1, & \text{if pixel}(x, y) \text{ is in region} \\ 0, & \text{otherwise} \end{cases} \quad (6.6)$$

This two-dimensional array can be easily accessed, unioned, merged or intersected by AND/OR logic operations. However it requires much space, and also does not represent the boundary of the region in a useful form.

Y-axis: A more compact representation than the above is Y-axis representation, which also allows all the above operations. Here each pixel

row (Y) belonging to the region is encoded as a list of X coordinates of pixel going in and out of the region, i.e. $((y, x_{in}, x_{out}))$

Example:



The Y-representation becomes inefficient when the region is long and thin along y axis (why?). In this case X-axis representation can be used.

Quad Tree: This representation is useful for the encoding occupancy. Each node of the tree has 4 children. The children represent regions. For example in the region shown in Fig. 6.10b, the quad tree is given in Fig. 6.10a. The gray nodes represent the mixed regions which must be split further. The black nodes represent pixels on the region, and white nodes are outside the regions.

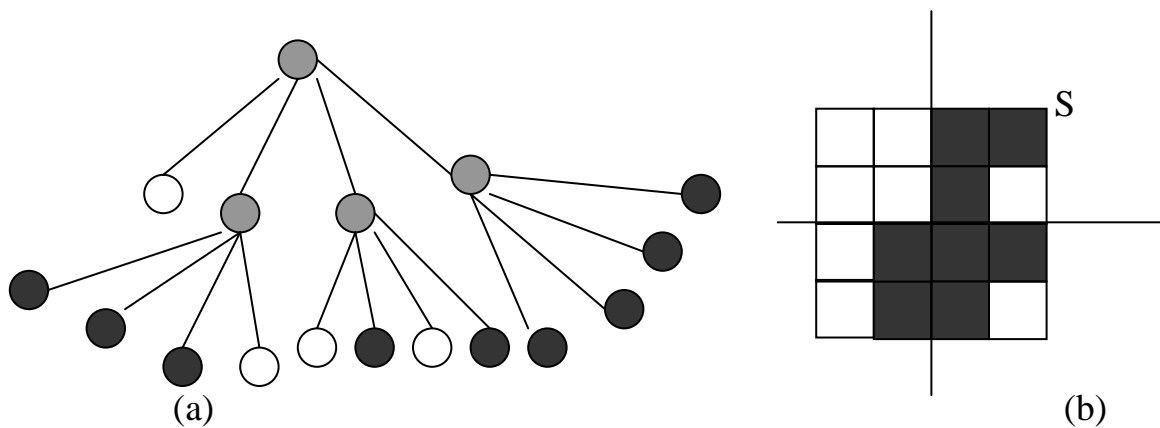


Fig. 6.10 Quad tree representation of a region.

Polylines: In this representation boundary of a region is approximated by a number of line segments and the coordinates of those segments are stored as

$\{(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)\}$. For example, the region in Fig 6.11 is approximated by six line segments, which are then stored by the coordinates of the intersections of the lines segments.

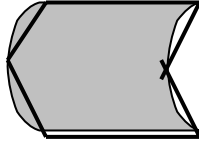


Fig. 6.11 polyline approximation of a region.

Any accuracy of representation can be achieved by increasing the number of line segments.

Chain Code: This is another representation, and is based on 4 or 8 directions (connectivity). A chain code can be generated by following the boundary in a clockwise direction and assigning directions to segments connecting pixels.

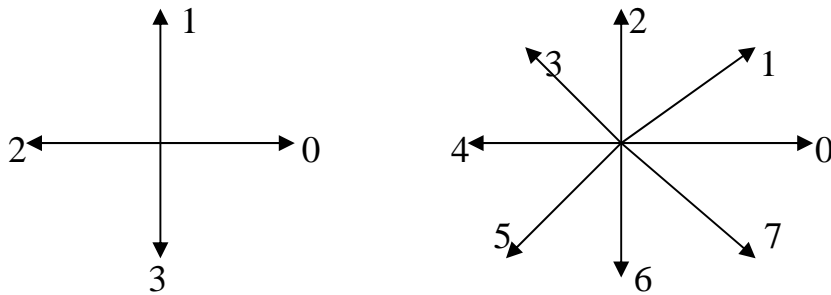


Fig. 6.12 4- and 8-Chain code

For example starting at point S in Fig. 6.10, and following the boundary clockwise using 4-directions, we have 2233233001012101.

If the region is long, the resulting codes will be very long. Also, small disturbances due to noise will cause changes to the boundary shape. To overcome these difficulties, we can resample the boundary by selecting a larger grid.

Fourier Descriptor: This method is used to reduce the number of points describing a boundary to a much smaller number without losing much precision. Suppose that the boundary of a region is represented by a set of point with their x-y coordinates, i.e.

$$P_k = (x_k, y_k), \quad k = 0, 1, \dots, N - 1$$

where N is the number of points (pixels on the boundary). Fourier descriptor treats each point as a complex number, i.e.

$$S_k = x_k + j y_k, \quad k = 0, 1, \dots, N - 1$$

A one dimensional Fourier transform is then applied to the S_k as follows

$$S(u) = \frac{1}{N} \sum_{k=0}^{N-1} S_k e^{-j2\pi uk / N}; \quad u = 0, 1, \dots, N - 1$$

Now we retain only $N_0 \ll N$ values of $S(u)$, and set the remain values to zero. In other words,

$$\hat{S}(u) = S(u); \quad u = 0, 1, \dots, N_0 - 1$$

which means that the high frequency contents of $S(u)$ are removed. This Fourier descriptor $\hat{S}(u)$ is then stored. Typically, $N_0 = 0.1 N$, and thus only a small number of points are stored. To get the boundary in the spatial domain, we perform the inverse Fourier transform on , i.e.

$$\hat{S}_k = \sum_{u=0}^{N_0-1} \hat{S}(u) e^{2\pi uk / N} \quad k = 0, 1, \dots, N-1$$

Note that the spatial representation still has N points, and is a good approximation of $S_k = x_k + j y_k, \quad k = 0, 1, \dots, N - 1$.

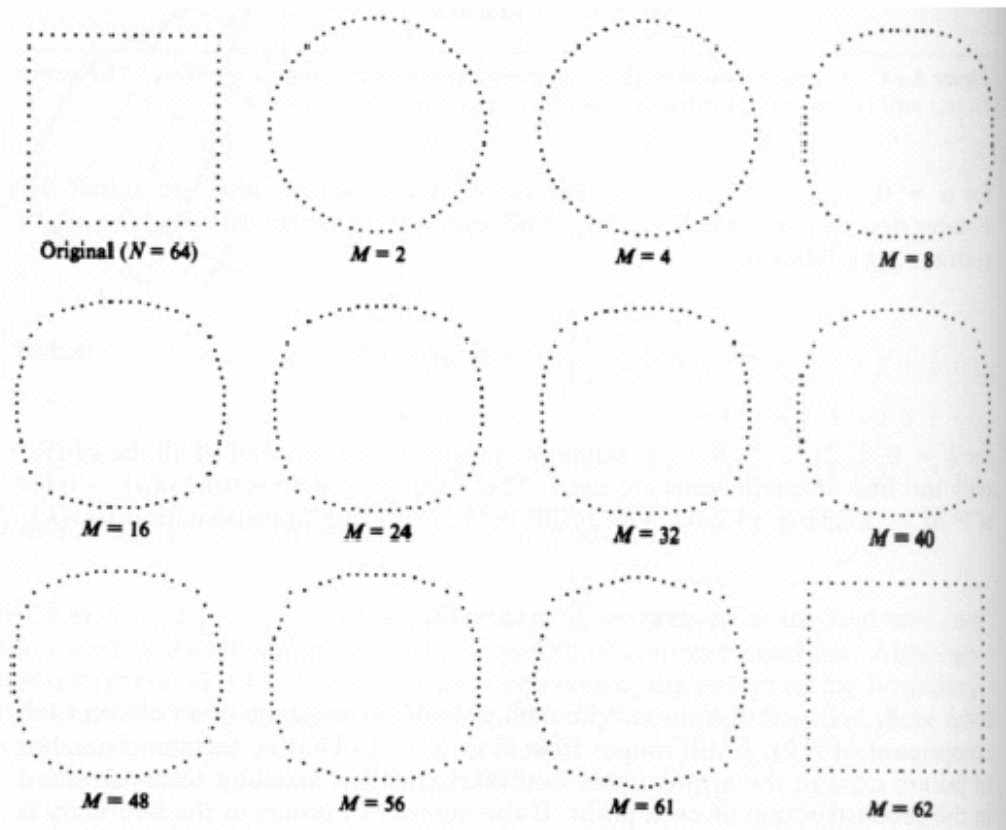


Figure 6: Examples of reconstructions from Fourier descriptors. M is the number of descriptors taken to reconstruct.

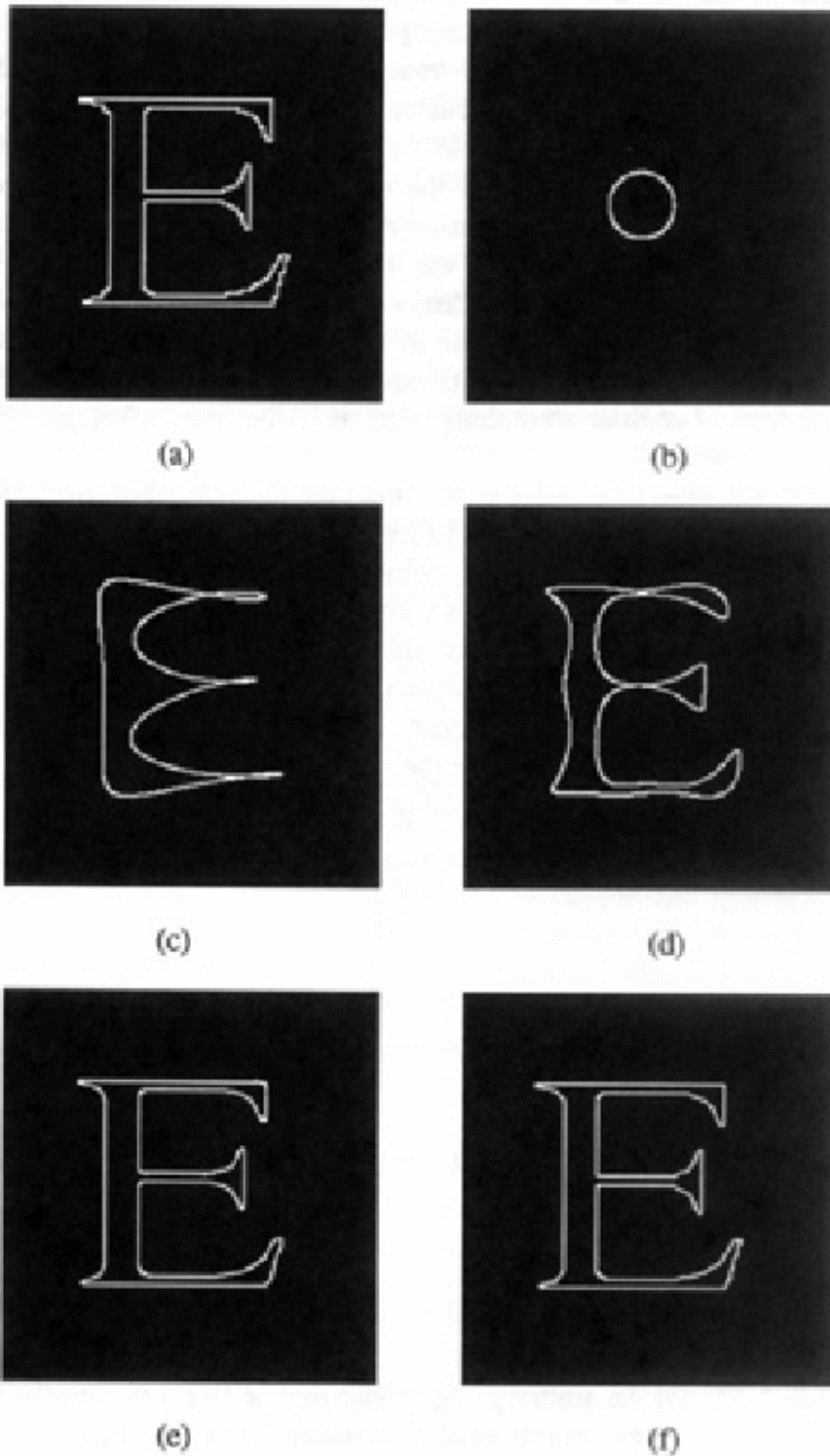


Figure 7: Examples using Fourier descriptors: (a) the original edge image with 1024 edge pixels, (b) 3 Fourier coefficients, (c) 21 Fourier coefficients, (d) 61 Fourier coefficients, (e) 201 Fourier coefficients, and (f) 401 Fourier coefficients.

6.6 Shape Measures of a Region

There are several measures that identify the shape and size of a region.

Area: The total number of pixels in a region is the area of that region. Area can be deduced from the region representation. For example, if the polyline consists of m points with coordinates $(x[i], y[i]), i=0,1, \dots, m-1$, then the area of the region is obtained from (why?)

$$A = \frac{1}{2} \sum_{i=0}^{m-1} (x[i+1]y[i] - x[i]y[i+1]) \quad (6.7)$$

Eccentricity: This is the ratio of maximum width to maximum height of a region as obtained from two perpendicular lines, i.e. $\eta = \frac{CD}{AB}$ as shown in Fig. 6.13.

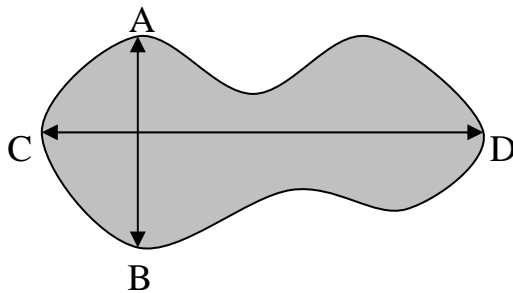


Fig. 6.13 Definition of eccentricity

Euler Number: The Euler number E is defined as

$$E = C - H \quad (6.8)$$

where C is the number of connected components of the region, and H is the number of holes in the region. The Euler numbers for the regions (a)-(c) of Fig. 6.14 are, $E = 1 - 2 = -1$, $E = 1 - 0 = 1$ and $E = 1 - 1 = 0$, respectively.

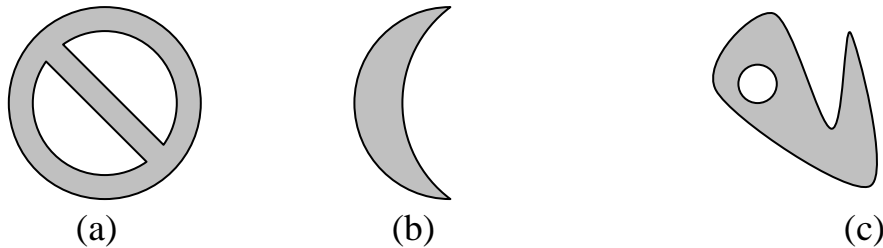


Fig. 6.14 Regions

Euler number describes a shape property that is unaffected by any deformation, so long as there is no tearing or joining. This type of deformation is sometime called rubber sheet distortion.

Signature: This is a one dimensional function giving a measure of a boundary. Given a binary image function $R(x,y)$ representing a region, then the horizontal and vertical signatures as defined, respectively, as

$$S(y) = \sum_x R(x,y) \quad \text{and} \quad S(x) = \sum_y R(x,y) \quad (6.9)$$

Centroid: The centroid of a function $f(x,y)$, which could be non-binary, is defined as

$$x_c = \frac{1}{A} \sum_{x=0}^{w-1} \sum_{y=0}^{h-1} x f(x,y); \quad y_c = \frac{1}{A} \sum_{x=0}^{w-1} \sum_{y=0}^{h-1} y f(x,y) \quad (6.10)$$

where A is the area, and w and h are the width and height of the image, respectively. For a binary image (region) $f(x,y) = 1$ if the pixel (x,y) is in the region, otherwise $f(x,y) = 0$.

Moments: These provide measure of distribution of values across an axis. The p and q moments are defined as

$$M_{pq} = \sum_{x=0}^{w-1} \sum_{y=0}^{h-1} x^p y^q f(x,y) \quad (6.11)$$

Note that:

$$\begin{aligned} M_{00} &= \sum_{x=0}^{w-1} \sum_{y=0}^{h-1} f(x,y) = \text{area} = A \\ M_{10} &= \sum_{x=0}^{w-1} \sum_{y=0}^{h-1} x f(x,y) = A x_c = M_{00} x_c \\ M_{01} &= \sum_{x=0}^{w-1} \sum_{y=0}^{h-1} y f(x,y) = A y_c = M_{00} y_c \end{aligned} \quad (6.12)$$

The central moment is defined as

$$\mu_{pq} = \sum_{x=0}^{w-1} \sum_{y=0}^{h-1} (x - x_c)^p (y - y_c)^q f(x, y) \quad (6.13)$$

That is the moments with the origin of axis located on the centroid.

Circularity: This is a measure of the roundness of a region. Let (x_i, y_i) be pixels on the boundary of a region and determine the distance to this pixel from the centroid as

$$d_i = \sqrt{(x - x_c)^2 + (y - y_c)^2} \quad (6.14)$$

If there are m points on the boundary, the average distance is

$$d_{ave} = \frac{1}{m} \sum_{i=0}^{m-1} d_i \quad (6.15)$$

Then the circularity is defined as the standard deviation, i.e.

$$\sigma = \sqrt{\sum_{i=0}^{m-1} (d_i - d_{ave})^2} \quad (6.16)$$

A perfectly circular region has $\sigma = 0$ (why?). Higher values of σ correspond to non-circular regions.

Rectangularity: This measure is defined as

$$\rho = \frac{A}{LW} \quad (6.17)$$

where A is the area of the region and L and W are the length and width of the rectangular bounding box. The maximum value of ρ is equal to 1.

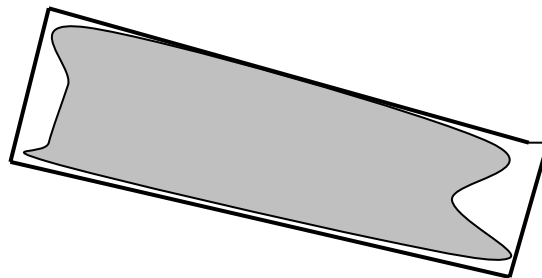


Fig. 6.15 Bounding box of a region.