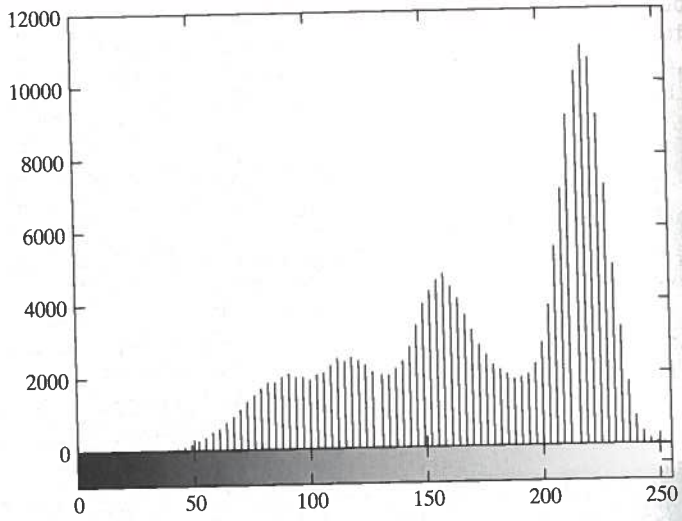


FIGURE 10.15 Histogram of Fig. 10.14(a).

10.4.3 Region Splitting and Merging

The procedure just discussed grows regions from a set of seed points. An alternative is to subdivide an image initially into a set of arbitrary, disjointed regions and then merge and/or split the regions in an attempt to satisfy the conditions stated in Section 10.4.1. The basics of splitting and merging are discussed next.

Let R represent the entire image region and select a predicate P. One approach for segmenting R is to subdivide it successively into smaller and smaller quadrant regions so that, for any region R_i, $P(R_i)$ = TRUE. We start with the entire region. If $P(R)$ = FALSE, we divide the image into quadrants. If P is FALSE for any quadrant, we subdivide that quadrant into subquadrants, and so on. This particular splitting technique has a convenient representation in the form of a so-called *quadtree*; that is, a tree in which each node has exactly four descendants, as illustrated in Fig. 10.16 (the subimages corresponding to the nodes of a quadtree sometimes are called *quadregions* or *quadimages*). Note that the root of the tree corresponds to the entire image and that each node corresponds to the subdivision of a node into four descendant nodes. In this case, only R_4 was subdivided further.

If only splitting is used, the final partition normally contains adjacent regions with identical properties. This drawback can be remedied by allowing merging, as

a b

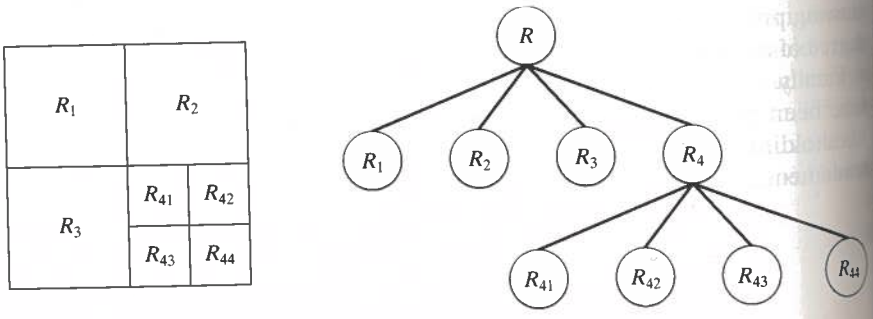
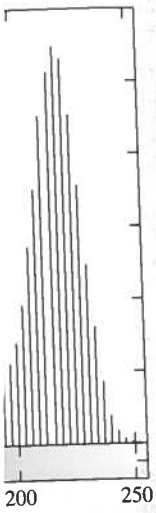
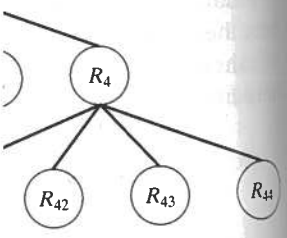


FIGURE 10.16 (a) Partitioned image. (b) Corresponding quadtree.



seed points. An alterna-
y, disjointed regions and
sfy the conditions stated
discussed next.
edicate P. One approach
ller and smaller quadrant
start with the entire re-
drants. If P is FALSE for
rants, and so on. This par-
tion in the form of a so-
actly four descendants, as
the nodes of a quadtree
e that the root of the tree
esponds to the subdivision
R_4 was subdivided further.
contains adjacent regions
ed by allowing merging, as



well as splitting. Satisfying the constraints of Section 10.4.1 requires merging only adjacent regions whose combined pixels satisfy the predicate P. That is, two adjacent regions R_j and R_k are merged only if $P(R_j \cup R_k) = \text{TRUE}$.

The preceding discussion may be summarized by the following procedure in which, at any step,

1. Split into four disjoint quadrants any region R_i for which $P(R_i) = \text{FALSE}$.
2. When no further splitting is possible, merge any adjacent regions R_j and R_k for which $P(R_j \cup R_k) = \text{TRUE}$.
3. Stop when no further merging is possible.

Numerous variations of the preceding basic theme are possible. For example, a significant simplification results if we allow merging of any two adjacent regions R_i and R_j if each one satisfies the predicate individually. This results in a much simpler (and faster) algorithm because testing of the predicate is limited to individual quadregions. As Example 10.9 shows, this simplification is still capable of yielding good segmentation results in practice. Using this approach in step 2 of the procedure, all quadregions that satisfy the predicate are filled with 1s and their connectivity can be easily examined using, for example, function `imreconstruct`. This function, in effect, accomplishes the desired merging of adjacent quadregions. The quadregions that do not satisfy the predicate are filled with 0s to create a segmented image.

The function in IPT for implementing quadtree decomposition is `qtdecomp`. The syntax of interest in this section is


```
S = qtdecomp(f, @split_test, parameters)
```

Other forms of `qtdecomp` *are discussed in Section 11.2.2.*

where `f` is the input image and `S` is a sparse matrix containing the quadtree structure. If `S(k, m)` is nonzero, then `(k, m)` is the upper-left corner of a block in the decomposition and the size of the block is `S(k, m)`. Function `split_test` (see function `splitmerge` below for an example) is used to determine whether a region is to be split or not, and `parameters` are any additional parameters (separated by commas) required by `split_test`. The mechanics of this are similar to those discussed in Section 3.4.2 for function `coltfilt`.

To get the actual quadregion pixel values in a quadtree decomposition we use function `qtgetblk`, with syntax


```
[vals, r, c] = qtgetblk(f, S, m)
```

where `vals` is an array containing the values of the blocks of size `m` × `m` in the quadtree decomposition of `f`, and `S` is the sparse matrix returned by `qtdecomp`. Parameters `r` and `c` are vectors containing the row and column coordinates of the upper-left corners of the blocks.

We illustrate the use of function `qtdecomp` by writing a basic split-and-merge M-function that uses the simplification discussed earlier, in which two regions are merged if each satisfies the predicate individually. The function, which we call `splitmerge`, has the following calling syntax:

```
g = splitmerge(f, mindim, @predicate)
```

where `f` is the input image and `g` is the output image in which each connected region is labeled with a different integer. Parameter `mindim` defines the size of the smallest block allowed in the decomposition; this parameter has to be a positive integer power of 2.

Function `predicate` is a user-defined function that must be included in the MATLAB path. Its syntax is

```
flag = predicate(region)
```

This function must be written so that it returns `true` (a logical 1) if the pixels in `region` satisfy the predicate defined by the code in the function; otherwise, the value of `flag` must be `false` (a logical 0). Example 10.9 illustrates the use of this function.

Function `splitmerge` has a simple structure. First, the image is partitioned using function `qtdecomp`. Function `split_test` uses `predicate` to determine if a region should be split or not. Because when a region is split into four it is not known which (if any) of the resulting four regions will pass the predicate test individually, it is necessary to examine the regions after the fact to see which regions in the partitioned image pass the test. Function `predicate` is used for this purpose also. Any quadregion that passes the test is filled with 1s. Any that does not is filled with 0s. A marker array is created by selecting one element of each region that is filled with 1s. This array is used in conjunction with the partitioned image to determine region connectivity (adjacency); function `imreconstruct` is used for this purpose.

The code for function `splitmerge` follows. The simple predicate function shown in the comments section of the code is used in Example 10.9. Note that the size of the input image is brought up to a square whose dimensions are the minimum integer power of 2 that encompasses the image. This is a requirement of function `qtdecomp` to guarantee that splits down to size 1 are possible.

`splitmerge`

```
function g = splitmerge(f, mindim, fun)
%SPLITMERGE Segment an image using a split-and-merge algorithm.
%   G = SPLITMERGE(F, MINDIM, @PREDICATE) segments image F by using a
%   split-and-merge approach based on quadtree decomposition. MINDIM
%   (a positive integer power of 2) specifies the minimum dimension
%   of the quadtree regions (subimages) allowed. If necessary, the
%   program pads the input image with zeros to the nearest square
%   size that is an integer power of 2. This guarantees that the
%   algorithm used in the quadtree decomposition will be able to
%   split the image down to blocks of size 1-by-1. The result is
%   cropped back to the original size of the input image. In the
%   output, G, each connected region is labeled with a different
%   integer.
%
%   Note that in the function call we use @PREDICATE for the value of
%   fun. PREDICATE is a function in the MATLAB path, provided by the
%   user. Its syntax is
%
```

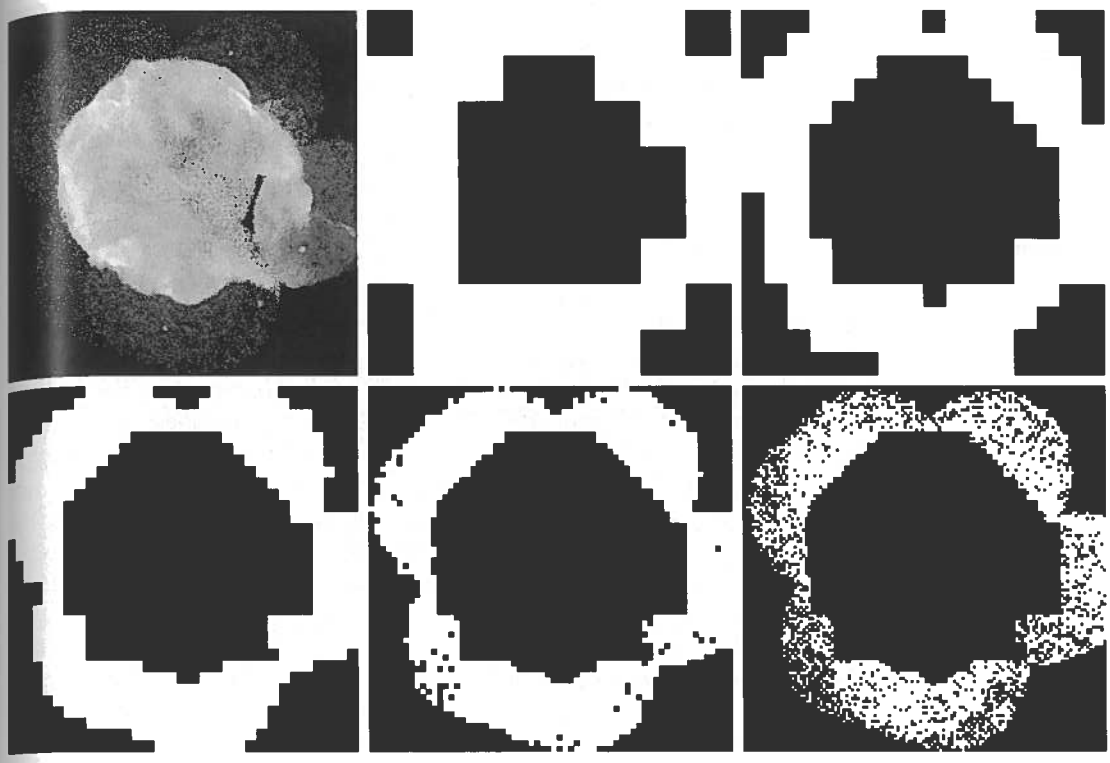
```
%     FLAG = PREDICATE(REGION) which must return TRUE if the pixels
%     in REGION satisfy the predicate defined by the code in the
%     function; otherwise, the value of FLAG must be FALSE.
%
%  The following simple example of function PREDICATE is used in
%  Example 10.9 of the book. It sets FLAG to TRUE if the
%  intensities of the pixels in REGION have a standard deviation
%  that exceeds 10, and their mean intensity is between 0 and 125.
%  Otherwise FLAG is set to false.
%
%     function flag = predicate(region)
%     sd = std2(region);
%     m = mean2(region);
%     flag = (sd > 10) & (m > 0) & (m < 125);

% Pad image with zeros to guarantee that function qtdecomp will
% split regions down to size 1-by-1.
Q = 2^nextpow2(max(size(f)));
[M, N] = size(f);
f = padarray(f, [Q - M, Q - N], 'post');

% Perform splitting first.
S = qtdecomp(f, @split_test, mindim, fun);

% Now merge by looking at each quadregion and setting all its
% elements to 1 if the block satisfies the predicate.

% Get the size of the largest block. Use full because S is sparse.
Lmax = full(max(S(:)));
% Set the output image initially to all zeros. The MARKER array is
% used later to establish connectivity.
g = zeros(size(f));
MARKER = zeros(size(f));
% Begin the merging stage.
for K = 1:Lmax
  [vals, r, c] = qtgetblk(f, S, K);
  if ~isempty(vals)
     % Check the predicate for each of the regions
     % of size K-by-K with coordinates given by vectors
     % r and c.
     for I = 1:length(r)
        xlow = r(I); ylow = c(I);
        xhigh = xlow + K - 1; yhigh = ylow + K - 1;
        region = f(xlow:xhigh, ylow:yhigh);
        flag = feval(fun, region);
        if flag
           g(xlow:xhigh, ylow:yhigh) = 1;
           MARKER(xlow, ylow) = 1;
        end
     end
  end
end
```

`feval(fun, param)` *evaluates function* `fun` *with parameter* `param`. *See the help page for* `feval` *for other syntax forms applicable to this function.*

```
% Finally, obtain each connected region and label it with a
% different integer value using function bwlabel.
g = bwlabel(imreconstruct(MARKER, g));

% Crop and exit
g = g(1:M, 1:N);

%-----------------------------------------------------------------%
function v = split_test(B, mindim, fun)
% THIS FUNCTION IS PART OF FUNCTION SPLIT-MERGE. IT DETERMINES
% WHETHER QUADREGIONS ARE SPLIT. The function returns in v
% logical 1s (TRUE) for the blocks that should be split and
% logical 0s (FALSE) for those that should not.

% Quadregion B, passed by qtdecomp, is the current decomposition of
% the image into k blocks of size m-by-m.

% k is the number of regions in B at this point in the procedure.
k = size(B, 3);

% Perform the split test on each block. If the predicate function
% (fun) returns TRUE, the region is split, so we set the appropriate
% element of v to TRUE. Else, the appropriate element of v is set to
% FALSE.
v(1:k) = false;
for I = 1:k
   quadregion = B(:, :, I);
   if size(quadregion, 1) <= mindim
      v(I) = false;
      continue
   end
   flag = feval(fun, quadregion);
   if flag
      v(I) = true;
   end
end
```

EXAMPLE 10.9: Image segmentation using region splitting and merging.

■ Figure 10.17(a) shows an X-ray band image of the Cygnus Loop. The image is of size 256 × 256 pixels. The objective of this example is to segment out of the image the "ring" of less dense matter surrounding the dense center. The region of interest has some obvious characteristics that should help in its segmentation. First, we note that the data has a random nature to it, indicating that its standard deviation should be greater than the standard deviation of the background (which is 0) and of the large central region. Similarly, the mean value (average intensity) of a region containing data from the outer ring should be greater than the mean of the background (which is 0) and less than the mean of the large, lighter central region. Thus, we should be able to segment the region of interest by using these two parameters. In fact, the predicate function shown as an example in the documentation of function `splitmerge` contains this knowledge about the problem. The parameters were determined by computing the mean and standard deviation of various regions in Fig. 10.17(a).



a b c
d e f

FIGURE 10.17 Image segmentation by a split-and-merge procedure. (a) Original image. (b) through (f) results of segmentation using function `splitmerge` with values of `mindim` equal to 32, 16, 8, 4, and 2, respectively. (Original image courtesy of NASA.)

Figures 10.17(b) through (f) show the results of segmenting Fig. 10.17(a) using function `splitmerge` with `mindim` values of 32, 16, 8, 4, and 2, respectively. All images show segmentation results with levels of detail that are inversely proportional to the value of `mindim`.

All results in Fig. 10.17 are reasonable segmentations. If one of these images were to be used as a mask to extract the region of interest out of the original image, then the result in Fig. 10.17(d) would be the best choice because it is the solid region with the most detail. An important aspect of the method just illustrated is its ability to "capture" in function `predicate` information about a problem domain that can help in segmentation. ■

10.5 Segmentation Using the Watershed Transform

In geography, a *watershed* is the ridge that divides areas drained by different river systems. A *catchment basin* is the geographical area draining into a river or reservoir. The *watershed transform* applies these ideas to gray-scale image processing in a way that can be used to solve a variety of image segmentation problems.