Digital Image Processing

BB1603391 116033910045 修宇亮

Problem 10 Requirement

10. Image representation and description

(a). Develop a program to implement the boundary following algorithm, the resampling grid and calculate the chain code and the first difference chain code. Use the image 'noisy_stroke.tif' for test. (For technique details, please refer to pp.818-822 (3rd edition, Gonzalez DIP) or boundaryfollowing.pdf at the same address of the slides.)

(b). Develop a program to implement the image description by the principal components (PC). Calculate and display the PC images and the reconstructed images from 2 PCs. Use the six images in 'washingtonDC.rar' as the test images.

Boundari Following

main.m

MATLAB

```
clear;clc;
img = mat2gray(imread('noisy_stroke.tif'));
ave_img = imfilter(img,ones(9,9)/81);
otsu_thresh = graythresh(ave_img);
otsu_img = im2bw(ave_img,otsu_thresh);
se = strel('square',3);
eroded = imerode(otsu_img,se);
boundary_img = im2bw(imsubtract(otsu_img,imfill(eroded,'holes')),0);
[row,col] = find(boundary_img==1);
[s,su] = bsubsamp([row col],50);

subsample = zeros(570,570);
for i=1:32
    subsample(s(i,1),s(i,2))=1;
end

[x,y] = clockwise_order(s(:,1),s(:,2));
c = connectpoly(x,y);
lines = zeros(570,570);
for i=1:1664
    lines(c(i,1),c(i,2))=1;
end

[m,n] = clockwise_order(su(:,1),su(:,2));
cn_result = fchcode([m,n],8);

subplot(2,3,1),imshow(img);title('original');
subplot(2,3,2),imshow(ave_img);title('imfilter ave');
subplot(2,3,3),imshow(otsu_img);title('otsu');
subplot(2,3,4),imshow(boundary_img);title('boundary');
subplot(2,3,5),imshow(subsample);title('subsample');
subplot(2,3,6),imshow(lines);title('lines');
```

bsubsamp.m

MATLAB

```
function [s, su] = bsubsamp(b, gridsep)
%BSUBSAMP Subsample a boundary.
%   [S, SU] = BSUBSAMP(B, GRIDSEP) subsamples the boundary B by
%   assigning each of its points to the grid node to which it is
%   closest.  The grid is specified by GRIDSEP, which is the
%   separation in pixels between the grid lines. For example, if
%   GRIDSEP = 2, there are two pixels in between grid lines. So, for
%   instance, the grid points in the first row would be at (1,1),
%   (1,4), (1,6), ..., and similarly in the y direction. The value
%   of GRIDSEP must be an even integer. The boundary is specified by
%   a set of coordinates in the form of an np-by-2 array.  It is
```

```
%   assumed that the boundary is one pixel thick.
%
%   Output S is the subsampled boundary. Output SU is normalized so
%   that the grid separation is unity.  This is useful for obtaining
%   the Freeman chain code of the subsampled boundary.

%   Copyright 2002-2004 R. C. Gonzalez, R. E. Woods, & S. L. Eddins
%   Digital Image Processing Using MATLAB, Prentice-Hall, 2004
%   $Revision: 1.8 $  $Date: 2004/11/04 20:17:59 $

% Check input.
[np, nc] = size(b);
if np < nc
   error('B must be of size np-by-2.');
end
if gridsep/2 ~= round(gridsep/2)
   error('GRIDSEP must be an even integer.')
end

% Some boundary tracing programs, such as boundaries.m, end with
% the beginning, resulting in a sequence in which the coordinates
% of the first and last points are the same. If this is the case
% in b, eliminate the last point.
if isequal(b(1, :), b(np, :))
   np = np - 1;
   b = b(1:np, :);
end

% Find the max x and y spanned by the boundary.
xmax = max(b(:, 1));
ymax = max(b(:, 2));

% Determine the integral number of grid lines with gridsep points in
% between them that encompass the intervals [1,xmax], [1,ymax].
GLx = ceil((xmax + gridsep)/(gridsep + 1));
GLy = ceil((ymax + gridsep)/(gridsep + 1));

% Form vectors of x and y grid locations.
I = 1:GLx;
% Vector of grid line locations intersecting x-axis.
X(I) = gridsep*I + (I - gridsep);

J = 1:GLy;
% Vector of grid line locations intersecting y-axis.
Y(J) = gridsep*J + (J - gridsep);

% Compute both components of the cityblock distance between each
% element of b and all the grid-line intersections.  Assign each
```

```
% point to the grid location for which each comp of the cityblock
% distance was <= gridsep/2. Note the use of meshgrid to
% optimize the code. Keep in mind that meshgrid requires that columns
% be listed first (see Chapter 2).
DIST = gridsep/2;
[YG, XG] = meshgrid(Y, X);
Q = 1;
for k=1:np
   [I,J] = find(abs(XG - b(k, 1)) <= DIST & abs(YG - b(k, 2)) <= ...
                DIST);
   % If point b(k,:) is equidistant from two or more grid intersections,
   % assign the point arbitrarily to the first one:
   I = I(1);
   J = J(1);
   ord = k; % To keep track of order of input coordinates.
   d1(Q, :) = cat(2, Y(J), ord);
   d2(Q, :) = cat(2, X(I), ord);
   Q = Q + 1;
end

% d is the set of points assigned to the new grid with line
% separation of gridsep. Note that it is formed as d=(d2,d1) to
% compensate for the coordinate transposition inherent in using
% meshgrid (see Chapter 2).
d = cat(2, d2(:, 1), d1); % The second column of d1 & d2 is ord.

% Sort the points using the values in ord, which is the last col in
% d.
d = fliplr(d); % So the last column becomes first.
d = sortrows(d);
d = fliplr(d); % Flip back.

% Eliminate duplicate rows in the first two components of
% d to create the output. The cw or ccw order MUST be preserved.
s = d(:, 1:2);
[s, m, n] = unique(s, 'rows');

% Function unique sorts the data--Restore to original order
% by using the contents of m.
s = [s, m];
s = fliplr(s);
s = sortrows(s);
s = fliplr(s);
s = s(:, 1:2);

% Scale to unit grid so that can use directly to obtain Freeman
% chain code.  The shape does not change.
su = round(s./gridsep) + 1;
```

clockwise_order.m

MATLAB

```
function [x y] = clockwise_order(x,y)
    cx = mean(x);
    cy = mean(y);
    a = atan2(y - cy, x - cx);
    [~, order] = sort(a);
    x = x(order);
    y = y(order);
end
```

connectpoly.m

MATLAB

```
function c = connectpoly(x, y)
%CONNECTPOLY Connects vertices of a polygon.
%   C = CONNECTPOLY(X, Y) connects the points with coordinates given
%   in X and Y with straight lines. These points are assumed to be a
%   sequence of polygon vertices organized in the clockwise or
%   counterclockwise direction. The output, C, is the set of points
%   along the boundary of the polygon in the form of an nr-by-2
%   coordinate sequence in the same direction as the input. The last
%   point in the sequence is equal to the first.

%   Copyright 2002-2004 R. C. Gonzalez, R. E. Woods, & S. L. Eddins
%   Digital Image Processing Using MATLAB, Prentice-Hall, 2004
%   $Revision: 1.4 $  $Date: 2003/11/21 14:29:16 $

v = [x(:), y(:)];

% Close polygon.
if ~isequal(v(end, :), v(1, :))
   v(end + 1, :) = v(1, :);
end

% Connect vertices.
segments = cell(1, length(v) - 1);
for I = 2:length(v)
   [x, y] = intline(v(I - 1, 1), v(I, 1), v(I - 1, 2), v(I, 2));
   segments{I - 1} = [x, y];
end

c = cat(1, segments{:});
```

intline.m

MATLAB

```
function [x, y] = intline(x1, x2, y1, y2)
%INTLINE Integer-coordinate line drawing algorithm.
%  [X, Y] = INTLINE(X1, X2, Y1, Y2) computes an
%  approximation to the line segment joining (X1, Y1) and
%  (X2, Y2) with integer coordinates.  X1, X2, Y1, and Y2
%  should be integers.  INTLINE is reversible; that is,
%  INTLINE(X1, X2, Y1, Y2) produces the same results as
%  FLIPUD(INTLINE(X2, X1, Y2, Y1)).

%   Copyright 1993-2002 The MathWorks, Inc.  Used with permission.
%   $Revision: 1.4 $  $Date: 2003/11/21 14:38:20 $

dx = abs(x2 - x1);
dy = abs(y2 - y1);

% Check for degenerate case.
if ((dx == 0) & (dy == 0))
   x = x1;
   y = y1;
  return;
end

flip = 0;
if (dx >= dy)
   if (x1 > x2)
      % Always "draw" from left to right.
      t = x1; x1 = x2; x2 = t;
      t = y1; y1 = y2; y2 = t;
      flip = 1;
   end
   m = (y2 - y1)/(x2 - x1);
   x = (x1:x2).';
   y = round(y1 + m*(x - x1));
else
   if (y1 > y2)
      % Always "draw" from bottom to top.
      t = x1; x1 = x2; x2 = t;
      t = y1; y1 = y2; y2 = t;
      flip = 1;
   end
   m = (x2 - x1)/(y2 - y1);
   y = (y1:y2).';
   x = round(x1 + m*(y - y1));
end

```

```
if (flip)
   x = flipud(x);
   y = flipud(y);
end
```

fchcode.m

MATLAB

```
function  c = fchcode(b, conn, dir)
%FCHCODE Computes the Freeman chain code of a boundary.
%   C = FCHCODE(B) computes the 8-connected Freeman chain code of a
%   set of 2-D coordinate pairs contained in B, an np-by-2 array. C
%   is a structure with the following fields:
%
%     c.fcc    = Freeman chain code (1-by-np)
%     c.diff   = First difference of code c.fcc (1-by-np)
%     c.mm     = Integer of minimum magnitude from c.fcc (1-by-np)
%     c.diffmm = First difference of code c.mm (1-by-np)
%     c.x0y0   = Coordinates where the code starts (1-by-2)
%
%   C = FCHCODE(B, CONN) produces the same outputs as above, but
%   with the code connectivity specified in CONN. CONN can be 8 for
%   an 8-connected chain code, or CONN can be 4 for a 4-connected
%   chain code. Specifying CONN=4 is valid only if the input
%   sequence, B, contains transitions with values 0, 2, 4, and 6,
%   exclusively.
%
%   C = FHCODE(B, CONN, DIR) produces the same outputs as above, but,
%   in addition, the desired code direction is specified. Values for
%   DIR can be:
%
%     'same'      Same as the order of the sequence of points in b.
%                 This is the default.
%
%     'reverse'   Outputs the code in the direction opposite to the
%                 direction of the points in B.  The starting point
%                 for each DIR is the same.
%
%   The elements of B are assumed to correspond to a 1-pixel-thick,
%   fully-connected, closed boundary. B cannot contain duplicate
%   coordinate pairs, except in the first and last positions, which
%   is a common feature of boundary tracing programs.
%
%   FREEMAN CHAIN CODE REPRESENTATION
%   The table on the left shows the 8-connected Freeman chain codes
%   corresponding to allowed deltax, deltay pairs. An 8-chain is
%   converted to a 4-chain if (1) if conn = 4; and (2) only
```

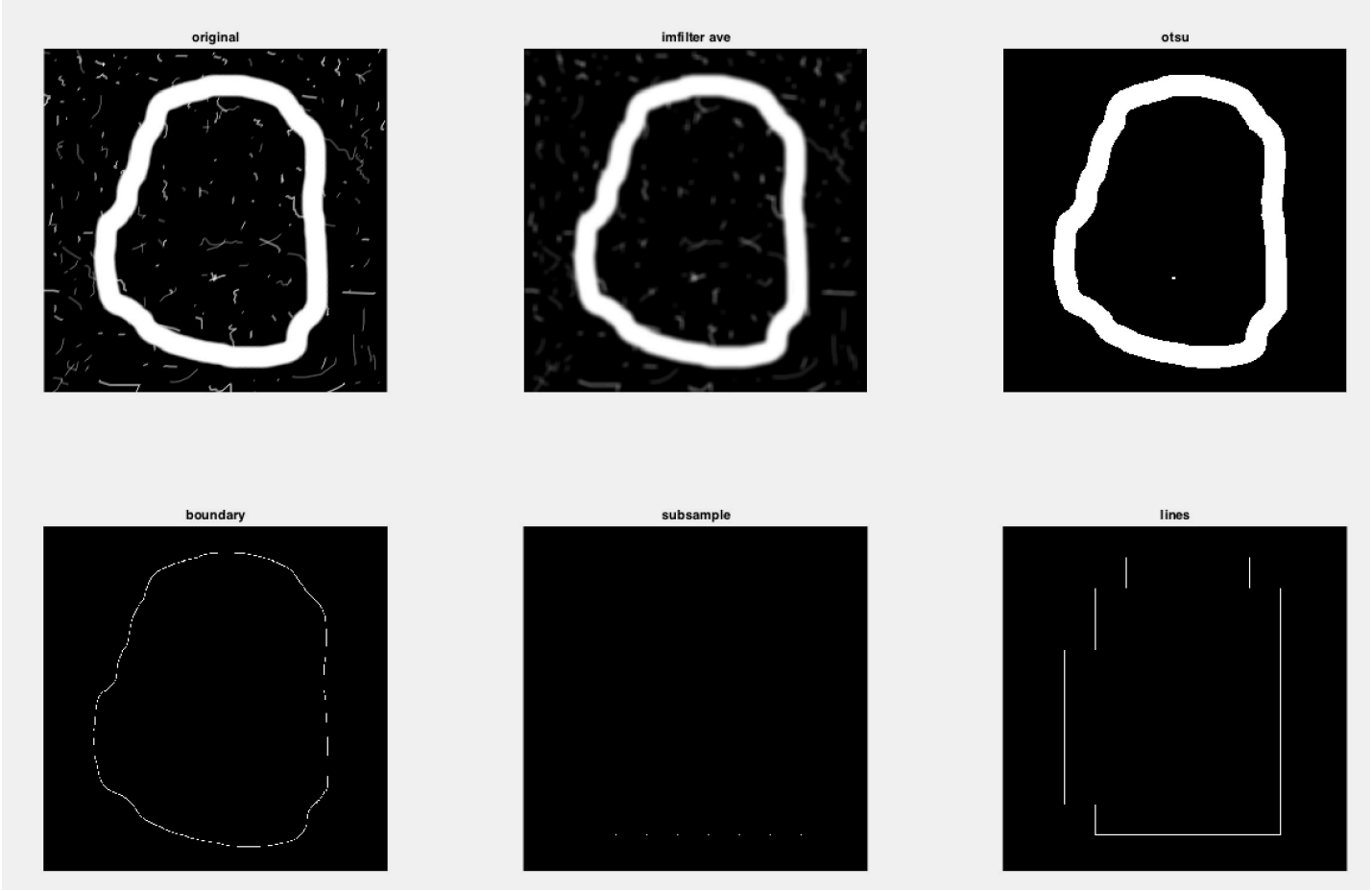
```
%   transitions 0, 2, 4, and 6 occur in the 8-code.  Note that
%   dividing 0, 2, 4, and 6 by 2 produce the 4-code.
%
%       ----------------------  ---------------
%       deltax | deltay | 8-code  corresp 4-code
%       ----------------------  ---------------
%         0        1       0            0
%        -1        1       1
%        -1        0       2            1
%        -1       -1       3
%         0       -1       4            2
%         1       -1       5
%         1        0       6            3
%         1        1       7
%       ----------------------  ---------------
%
%   The formula z = 4*(deltax + 2) + (deltay + 2) gives the following
%   sequence corresponding to rows 1-8 in the preceding table: z =
%   11,7,6,5,9,13,14,15. These values can be used as indices into the
%   table, improving the speed of computing the chain code. The
%   preceding formula is not unique, but it is based on the smallest
%   integers (4 and 2) that are powers of 2.

%   Copyright 2002-2004 R. C. Gonzalez, R. E. Woods, & S. L. Eddins
%   Digital Image Processing Using MATLAB, Prentice-Hall, 2004
%   $Revision: 1.6 $  $Date: 2003/11/21 14:34:49 $

% Preliminaries.
if nargin == 1
   dir = 'same';
   conn = 8;
elseif nargin == 2
   dir = 'same';
elseif nargin == 3
   % Nothing to do here.
else
   error('Incorrect number of inputs.')
end
[np, nc] = size(b);
if np < nc
   error('B must be of size np-by-2.');
end

% Some boundary tracing programs, such as boundaries.m, output a
% sequence in which the coordinates of the first and last points are
% the same. If this is the case, eliminate the last point.
if isequal(b(1, :), b(np, :))
   np = np - 1;
```

```
   b = b(1:np, :);
end

% Build the code table using the single indices from the formula
% for z given above:
C(11)=0; C(7)=1; C(6)=2; C(5)=3; C(9)=4;
C(13)=5; C(14)=6; C(15)=7;

% End of Preliminaries.

% Begin processing.
x0 = b(1, 1);
y0 = b(1, 2);
c.x0y0 = [x0, y0];

% Make sure the coordinates are organized sequentially:
% Get the deltax and deltay between successive points in b. The
% last row of a is the first row of b.
a = circshift(b, [-1, 0]);

% DEL = a - b is an nr-by-2 matrix in which the rows contain the
% deltax and deltay between successive points in b. The two
% components in the kth row of matrix DEL are deltax and deltay
% between point (xk, yk) and (xk+1, yk+1).  The last row of DEL
% contains the deltax and deltay between (xnr, ynr) and (x1, y1),
% (i.e., between the last and first points in b).
DEL = a - b;

% If the abs value of either (or both) components of a pair
% (deltax, deltay) is greater than 1, then by definition the curve
% is broken (or the points are out of order), and the program
% terminates.
if any(abs(DEL(:, 1)) > 1) | any(abs(DEL(:, 2)) > 1);
   error('The input curve is broken or points are out of order.')
end

% Create a single index vector using the formula described above.
z = 4*(DEL(:, 1) + 2) + (DEL(:, 2) + 2);

% Use the index to map into the table. The following are
% the Freeman 8-chain codes, organized in a 1-by-np array.
fcc = C(z);

% Check if direction of code sequence needs to be reversed.
if strcmp(dir, 'reverse')
   fcc = coderev(fcc); % See below for function coderev.
end
```

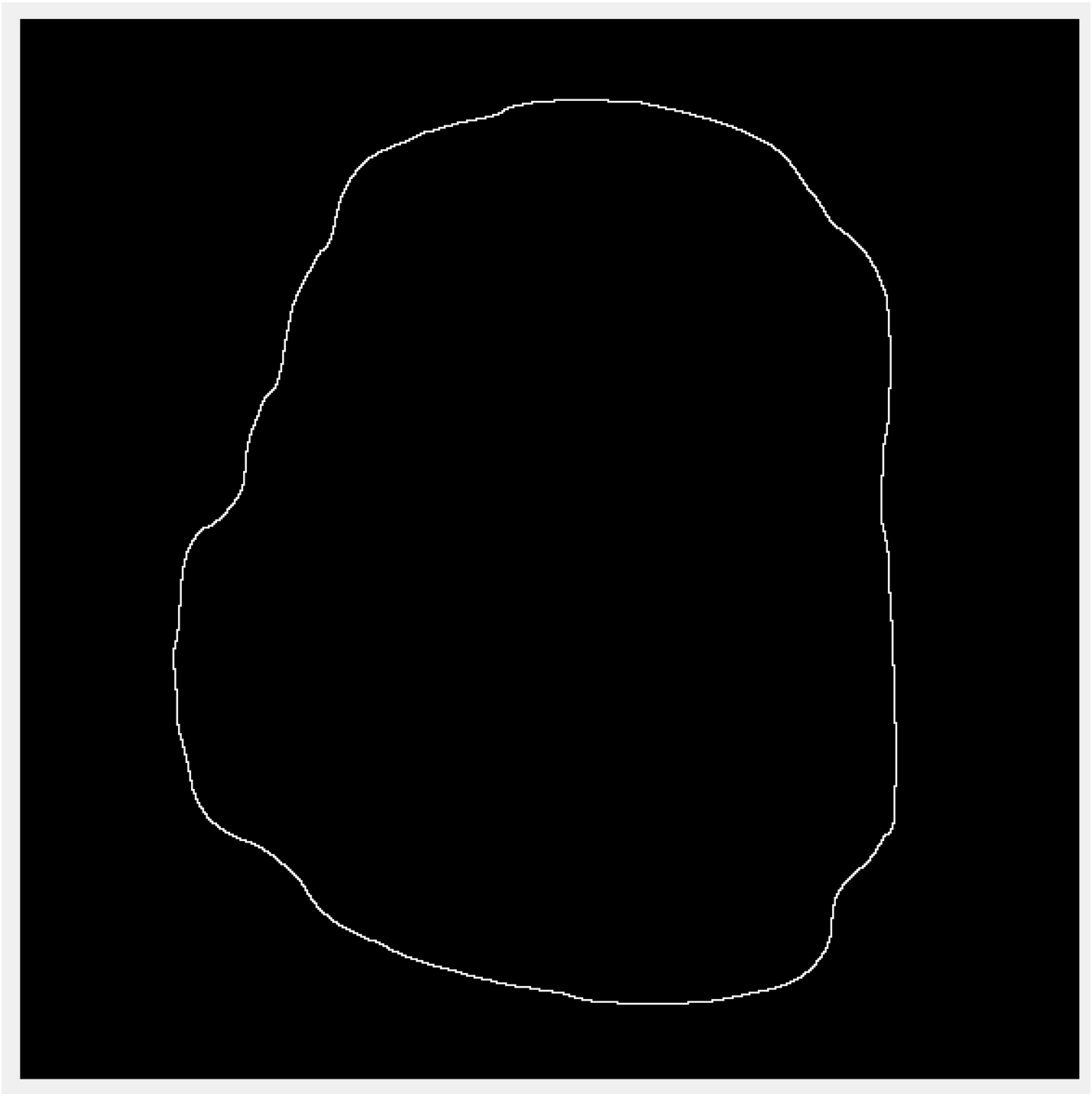
```
% If 4-connectivity is specified, check that all components
% of fcc are 0, 2, 4, or 6.
if conn == 4
   val = find(fcc == 1 | fcc == 3 | fcc == 5 | fcc ==7 );
   if isempty(val)
      fcc = fcc./2;
   else
      warning('The specified 4-connected code cannot be satisfied.')
   end
end

% Freeman chain code for structure output.
c.fcc = fcc;

% Obtain the first difference of fcc.
c.diff = codediff(fcc,conn); % See below for function codediff.

% Obtain code of the integer of minimum magnitude.
c.mm = minmag(fcc); % See below for function minmag.

% Obtain the first difference of fcc
c.diffmm = codediff(c.mm, conn);

%---------------------------------------------------------------------%
function cr = coderev(fcc)
%   Traverses the sequence of 8-connected Freeman chain code fcc in
%   the opposite direction, changing the values of each code
%   segment. The starting point is not changed. fcc is a 1-by-np
%   array.

% Flip the array left to right.  This redefines the starting point
% as the last point and reverses the order of "travel" through the
% code.
cr = fliplr(fcc);

% Next, obtain the new code values by traversing the code in the
% opposite direction. (0 becomes 4, 1 becomes 5, ... , 5 becomes 1,
% 6 becomes 2, and 7 becomes 3).
ind1 = find(0 <= cr & cr <= 3);
ind2 = find(4 <= cr & cr <= 7);
cr(ind1) = cr(ind1) + 4;
cr(ind2) = cr(ind2) - 4;

%---------------------------------------------------------------------%
function z = minmag(c)
%MINMAG Finds the integer of minimum magnitude in a chain code.
%   Z = MINMAG(C) finds the integer of minimum magnitude in a given
%   4- or 8-connected Freeman chain code, C. The code is assumed to
```

```
%   be a 1-by-np array.

% The integer of minimum magnitude starts with min(c), but there
% may be more than one such value. Find them all,
I = find(c == min(c));
% and shift each one left so that it starts with min(c).
J = 0;
A = zeros(length(I), length(c));
for k = I;
   J = J + 1;
   A(J, :) = circshift(c,[0 -(k-1)]);
end

% Matrix A contains all the possible candidates for the integer of
% minimum magnitude. Starting with the 2nd column, succesively find
% the minima in each column of A. The number of candidates decreases
% as the seach moves to the right on A.  This is reflected in the
% elements of J.  When length(J)=1, one candidate remains.  This is
% the integer of minimum magnitude.
[M, N] = size(A);
J = (1:M)';
for k = 2:N
   D(1:M, 1) = Inf;
   D(J, 1) = A(J, k);
   amin = min(A(J, k));
   J = find(D(:, 1) == amin);
   if length(J)==1
      z = A(J, :);
      return
   end
end

%--------------------------------------------------------------------%
function d = codediff(fcc, conn)
%CODEDIFF Computes the first difference of a chain code.
%   D = CODEDIFF(FCC) computes the first difference of code, FCC. The
%   code FCC is treated as a circular sequence, so the last element
%   of D is the difference between the last and first elements of
%   FCC.  The input code is a 1-by-np vector.
%
%   The first difference is found by counting the number of direction
%   changes (in a counter-clockwise direction) that separate two
%   adjacent elements of the code.

sr = circshift(fcc, [0, -1]); % Shift input left by 1 location.
delta = sr - fcc;
d = delta;
I = find(delta < 0);
```

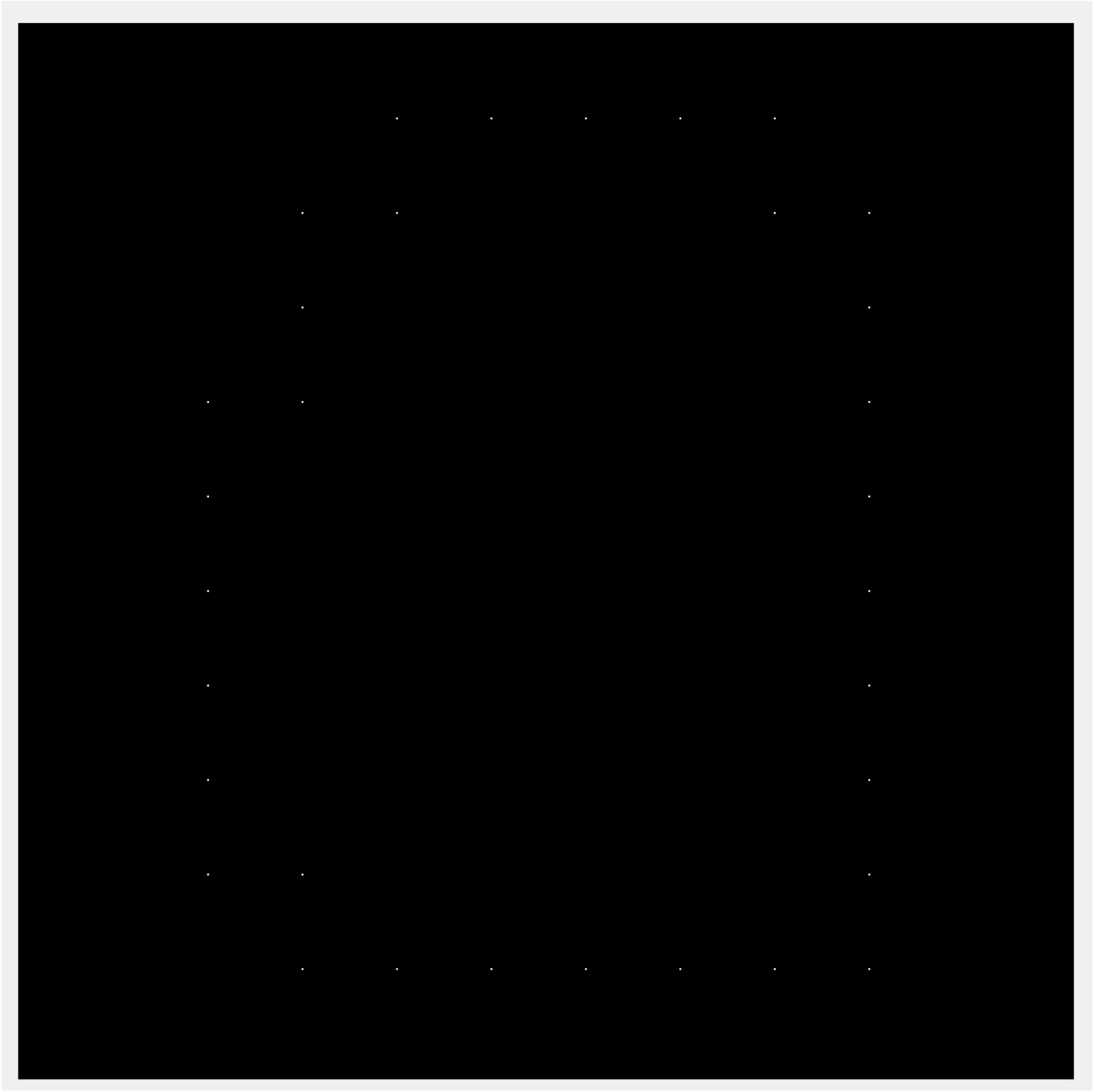
```
232
233 type = conn;
234 switch type
235 case 4 % Code is 4-connected
236    d(I) = d(I) + 4;
237 case 8 % Code is 8-connected
238    d(I) = d(I) + 8;
239 end
```



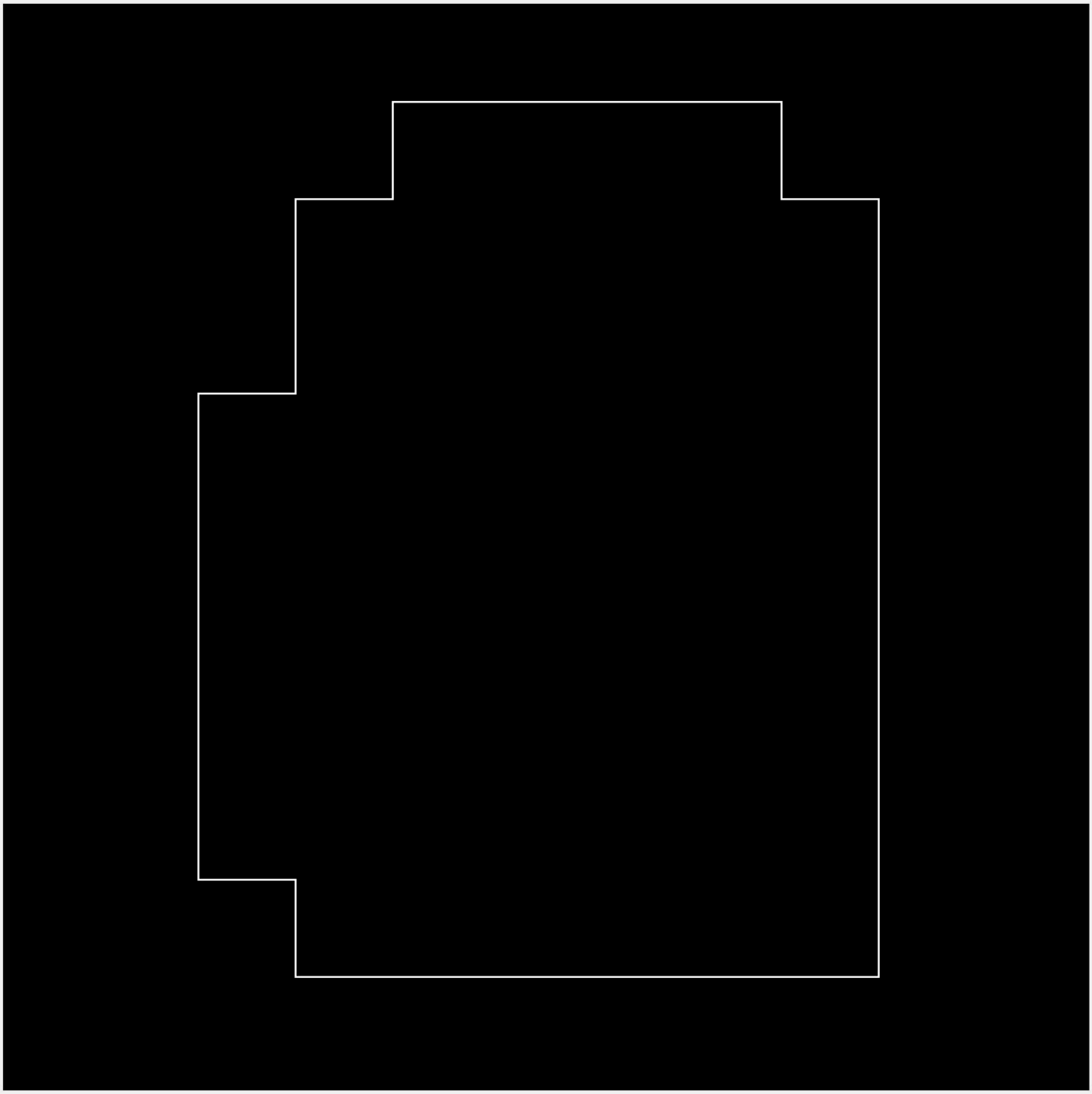
boundary image



subsample image



connected lines image



chain code result

MATLAB

```
>> cn_result.fcc

ans =

  Columns 1 through 15

     4     6     4     6     6     4     6     6     6     6     6     0     6     0

  Columns 16 through 30
```

```
    0     0     0     0     2     2     2     2     2     2     2     2     4     2

  Columns 31 through 32

    4     4

>> cn_result.diff

ans =

  Columns 1 through 15

    2     6     2     0     6     2     0     0     0     0     2     6     2     0

  Columns 16 through 30

    0     0     0     2     0     0     0     0     0     0     0     2     6     2

  Columns 31 through 32

    0     0

>> cn_result.mm

ans =

  Columns 1 through 15

    0     0     0     0     0     0     2     2     2     2     2     2     2     2

  Columns 16 through 30

    2     4     4     4     4     6     4     6     6     4     6     6     6     6

  Columns 31 through 32

    0     6

>> cn_result.diffmm

ans =

  Columns 1 through 15

    0     0     0     0     0     2     0     0     0     0     0     0     0     2

  Columns 16 through 30
```

```
     2     0     0     0     2     6     2     0     6     2     0     0     0     0

  Columns 31 through 32

     6     2

>>
```

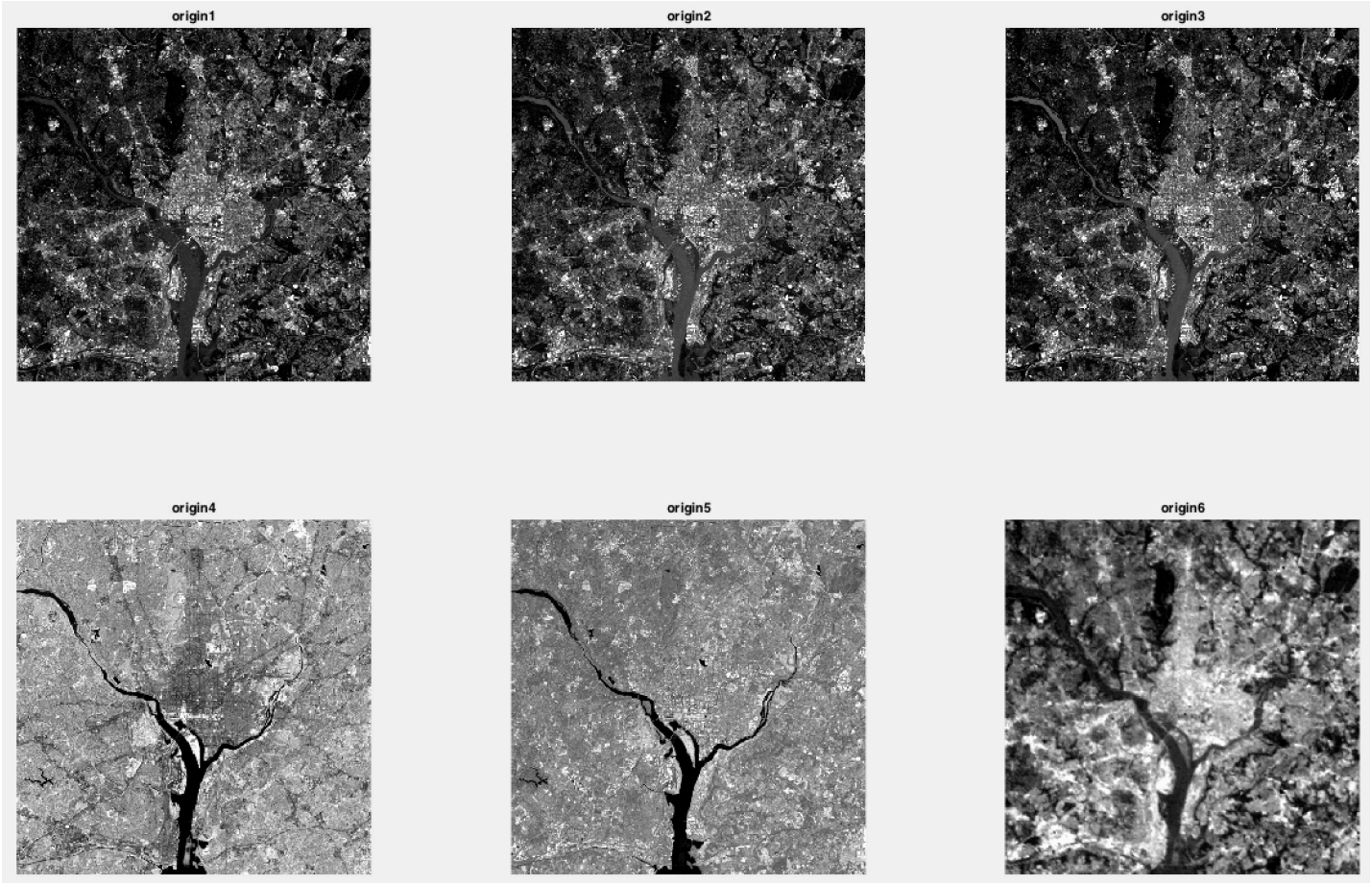
Principal Components

pc.m

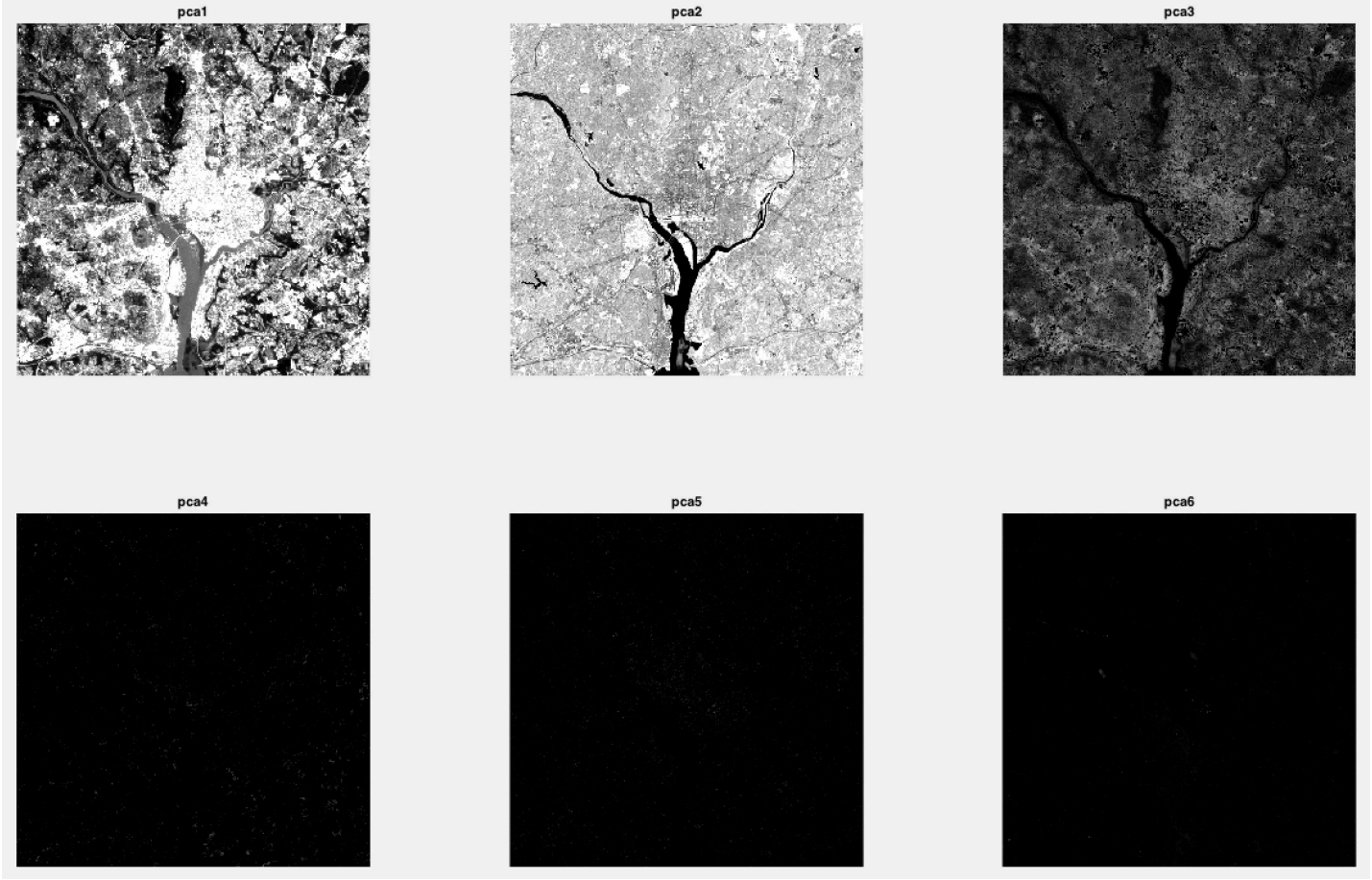
MATLAB

```matlab
clear;clc;
path = '/Users/Personals/Desktop/DIP/assignments/prob10/washingtonDC/';
comb_img = [];

for i=1:6
    wdc = double(imread([path 'WashingtonDC_Band' num2str(i) '.tif']));
    wdc_col = reshape(wdc,564*564,1);
    comb_img = [comb_img wdc_col];
end

comb_img = double(comb_img);
mu = mean(comb_img,2);
[coeff,score,latent] = pca(comb_img);
wdc_pcs = comb_img * coeff;
pca_result = zeros(564,564,6);
wdc_rec = bsxfun(@plus,score(:,1:2) * coeff(:,1:2)',mu);
rec_result = zeros(size(pca_result));
diff_result = zeros(size(pca_result));
comb_result = zeros(size(pca_result));

for i=1:6
    comb_result(:,:,i) = reshape(comb_img(:,i),564,564);
    pca_result(:,:,i) = reshape(wdc_pcs(:,i),564,564);
    rec_result(:,:,i) = reshape(wdc_rec(:,i),564,564);
    diff_result(:,:,i) = imsubtract(comb_result(:,:,i),rec_result(:,:,i));

    subplot(2,3,i),imshow(uint8(comb_result(:,:,i)));title(['origin' num2str(i)]);
    subplot(2,3,i),imshow(uint8(pca_result(:,:,i)));title(['pca' num2str(i)]);
    subplot(2,3,i),imshow(uint8(rec_result(:,:,i)));title(['rec' num2str(i)]);
    subplot(2,3,i),imshow(uint8(diff_result(:,:,i)));title(['diff' num2str(i)]);
end
```

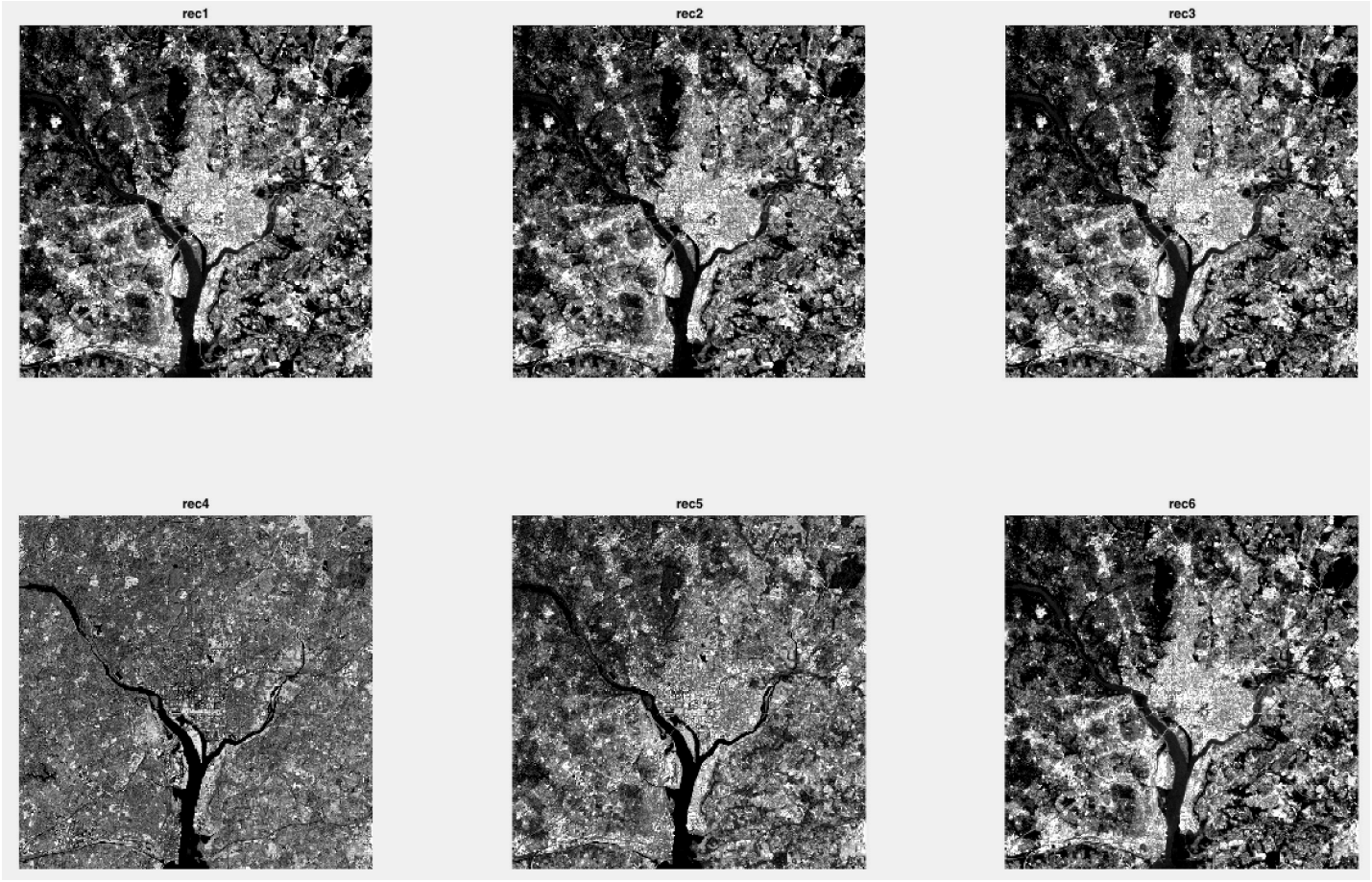
original image



principal components

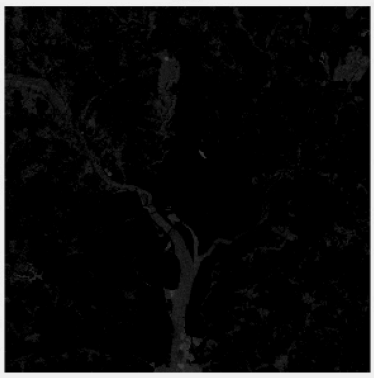


reconstruct image

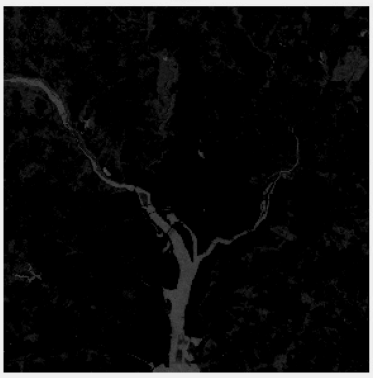


difference image

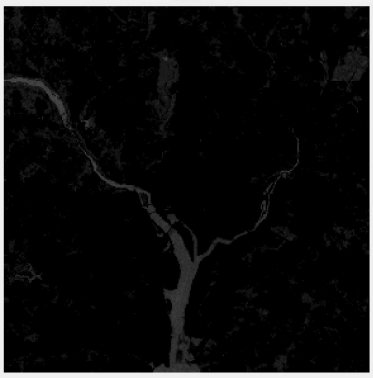
diff1



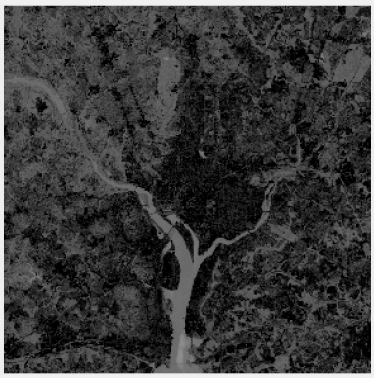
diff2



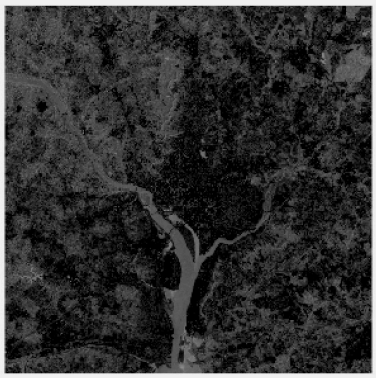
diff3



diff4



diff5



diff6