

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.

Boundary Following

main.m

```

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

11 |
12 | subsample = zeros(570,570);
13 | for i=1:32
14 |     subsample(s(i,1),s(i,2))=1;
15 | end

16 |
17 | [x,y] = clockwise_order(s(:,1),s(:,2));
18 | c = connectpoly(x,y);
19 | lines = zeros(570,570);
20 | for i=1:1664
21 |     lines(c(i,1),c(i,2))=1;
22 | end

23 |
24 | [m,n] = clockwise_order(su(:,1),su(:,2));
25 | cn_result = fchcode([m,n],8);

26 |
27 | subplot(2,3,1),imshow(img);title('original');
28 | subplot(2,3,2),imshow(ave_img);title('imfilter ave');
29 | subplot(2,3,3),imshow(otsu_img);title('otsu');
30 | subplot(2,3,4),imshow(boundary_img);title('boundary');
31 | subplot(2,3,5),imshow(subsample);title('subsample');
32 | subplot(2,3,6),imshow(lines);title('lines');

```

bsubsamp.m

```

1 | function [s, su] = bsubsamp(b, gridsep)
2 | %BSUBSAM Subsample a boundary.
3 | % [S, SU] = BSUBSAM(B, GRIDSEP) subsamples the boundary B by
4 | % assigning each of its points to the grid node to which it is
5 | % closest. The grid is specified by GRIDSEP, which is the
6 | % separation in pixels between the grid lines. For example, if
7 | % GRIDSEP = 2, there are two pixels in between grid lines. So, for
8 | % instance, the grid points in the first row would be at (1,1),
9 | % (1,4), (1,6), ..., and similarly in the y direction. The value
10 | % 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

```

```

11 % assumed that the boundary is one pixel thick.
12 %
13 % Output S is the subsampled boundary. Output SU is normalized so
14 % that the grid separation is unity. This is useful for obtaining
15 % the Freeman chain code of the subsampled boundary.
16
17 % Copyright 2002-2004 R. C. Gonzalez, R. E. Woods, & S. L. Eddins
18 % Digital Image Processing Using MATLAB, Prentice-Hall, 2004
19 % $Revision: 1.8 $ $Date: 2004/11/04 20:17:59 $
20
21 % Check input.
22 [np, nc] = size(b);
23 if np < nc
24     error('B must be of size np-by-2.');
25 end
26 if gridsep/2 ~= round(gridsep/2)
27     error('GRIDSEP must be an even integer.')
28 end
29
30 % Some boundary tracing programs, such as boundaries.m, end with
31 % the beginning, resulting in a sequence in which the coordinates
32 % of the first and last points are the same. If this is the case
33 % in b, eliminate the last point.
34 if isequal(b(1, :), b(np, :))
35     np = np - 1;
36     b = b(1:np, :);
37 end
38
39 % Find the max x and y spanned by the boundary.
40 xmax = max(b(:, 1));
41 ymax = max(b(:, 2));
42
43 % Determine the integral number of grid lines with gridsep points in
44 % between them that encompass the intervals [1,xmax], [1,ymax].
45 GLx = ceil((xmax + gridsep)/(gridsep + 1));
46 GLy = ceil((ymax + gridsep)/(gridsep + 1));
47
48 % Form vectors of x and y grid locations.
49 I = 1:GLx;
50 % Vector of grid line locations intersecting x-axis.
51 X(I) = gridsep*I + (I - gridsep);
52
53 J = 1:GLy;
54 % Vector of grid line locations intersecting y-axis.
55 Y(J) = gridsep*J + (J - gridsep);
56
57 % Compute both components of the cityblock distance between each
58 % element of b and all the grid-line intersections. Assign each

```

```

59 % point to the grid location for which each comp of the cityblock
60 % distance was <= gridsep/2. Note the use of meshgrid to
61 % optimize the code. Keep in mind that meshgrid requires that columns
62 % be listed first (see Chapter 2).
63 DIST = gridsep/2;
64 [YG, XG] = meshgrid(Y, X);
65 Q = 1;
66 for k=1:np
67     [I,J] = find(abs(XG - b(k, 1)) <= DIST & abs(YG - b(k, 2)) <= ...
68             DIST);
69     % If point b(k,:) is equidistant from two or more grid intersections,
70     % assign the point arbitrarily to the first one:
71     I = I(1);
72     J = J(1);
73     ord = k; % To keep track of order of input coordinates.
74     d1(Q, :) = cat(2, Y(J), ord);
75     d2(Q, :) = cat(2, X(I), ord);
76     Q = Q + 1;
77 end
78
79 % d is the set of points assigned to the new grid with line
80 % separation of gridsep. Note that it is formed as d=(d2,d1) to
81 % compensate for the coordinate transposition inherent in using
82 % meshgrid (see Chapter 2).
83 d = cat(2, d2(:, 1), d1); % The second column of d1 & d2 is ord.
84
85 % Sort the points using the values in ord, which is the last col in
86 % d.
87 d = fliplr(d); % So the last column becomes first.
88 d = sortrows(d);
89 d = fliplr(d); % Flip back.
90
91 % Eliminate duplicate rows in the first two components of
92 % d to create the output. The cw or ccw order MUST be preserved.
93 s = d(:, 1:2);
94 [s, m, n] = unique(s, 'rows');
95
96 % Function unique sorts the data--Restore to original order
97 % by using the contents of m.
98 s = [s, m];
99 s = fliplr(s);
100 s = sortrows(s);
101 s = fliplr(s);
102 s = s(:, 1:2);
103
104 % Scale to unit grid so that can use directly to obtain Freeman
105 % chain code. The shape does not change.
106 su = round(s./gridsep) + 1;

```

clockwise_order.m

MATLAB

```
1 function [x y] = clockwise_order(x,y)
2     cx = mean(x);
3     cy = mean(y);
4     a = atan2(y - cy, x - cx);
5     [~, order] = sort(a);
6     x = x(order);
7     y = y(order);
8 end
```

connectpoly.m

MATLAB

```
1 function c = connectpoly(x, y)
2 %CONNECTPOLY Connects vertices of a polygon.
3 % C = CONNECTPOLY(X, Y) connects the points with coordinates given
4 % in X and Y with straight lines. These points are assumed to be a
5 % sequence of polygon vertices organized in the clockwise or
6 % counterclockwise direction. The output, C, is the set of points
7 % along the boundary of the polygon in the form of an nr-by-2
8 % coordinate sequence in the same direction as the input. The last
9 % point in the sequence is equal to the first.
10 %
11 % Copyright 2002-2004 R. C. Gonzalez, R. E. Woods, & S. L. Eddins
12 % Digital Image Processing Using MATLAB, Prentice-Hall, 2004
13 % $Revision: 1.4 $ $Date: 2003/11/21 14:29:16 $
14 %
15 v = [x(:), y(:)];
16 %
17 % Close polygon.
18 if ~isequal(v(end, :), v(1, :))
19     v(end + 1, :) = v(1, :);
20 end
21 %
22 % Connect vertices.
23 segments = cell(1, length(v) - 1);
24 for I = 2:length(v)
25     [x, y] = intline(v(I - 1, 1), v(I, 1), v(I - 1, 2), v(I, 2));
26     segments{I - 1} = [x, y];
27 end
28 %
29 c = cat(1, segments{:});
```

intline.m

MATLAB

```
1 | function [x, y] = intline(x1, x2, y1, y2)
2 | %INTLINE Integer-coordinate line drawing algorithm.
3 | % [X, Y] = INTLINE(X1, X2, Y1, Y2) computes an
4 | % approximation to the line segment joining (X1, Y1) and
5 | % (X2, Y2) with integer coordinates. X1, X2, Y1, and Y2
6 | % should be integers. INTLINE is reversible; that is,
7 | % INTLINE(X1, X2, Y1, Y2) produces the same results as
8 | % FLIPUD(INTLINE(X2, X1, Y2, Y1)).
9 |
10| % Copyright 1993-2002 The MathWorks, Inc. Used with permission.
11| % $Revision: 1.4 $ $Date: 2003/11/21 14:38:20 $
12|
13| dx = abs(x2 - x1);
14| dy = abs(y2 - y1);
15|
16| % Check for degenerate case.
17| if ((dx == 0) & (dy == 0))
18|     x = x1;
19|     y = y1;
20|     return;
21| end
22|
23| flip = 0;
24| if (dx >= dy)
25|     if (x1 > x2)
26|         % Always "draw" from left to right.
27|         t = x1; x1 = x2; x2 = t;
28|         t = y1; y1 = y2; y2 = t;
29|         flip = 1;
30|     end
31|     m = (y2 - y1)/(x2 - x1);
32|     x = (x1:x2).';
33|     y = round(y1 + m*(x - x1));
34| else
35|     if (y1 > y2)
36|         % Always "draw" from bottom to top.
37|         t = x1; x1 = x2; x2 = t;
38|         t = y1; y1 = y2; y2 = t;
39|         flip = 1;
40|     end
41|     m = (x2 - x1)/(y2 - y1);
42|     y = (y1:y2).';
43|     x = round(x1 + m*(y - y1));
44| end
45|
```

```

46 if (flip)
47     x = flipud(x);
48     y = flipud(y);
49 end

```

fchcode.m

MATLAB

```

1 function c = fchcode(b, conn, dir)
2 %FCHCODE Computes the Freeman chain code of a boundary.
3 % C = FCHCODE(B) computes the 8-connected Freeman chain code of a
4 % set of 2-D coordinate pairs contained in B, an np-by-2 array. C
5 % is a structure with the following fields:
6 %
7 %     c.fcc      = Freeman chain code (1-by-np)
8 %     c.diff     = First difference of code c.fcc (1-by-np)
9 %     c.mm       = Integer of minimum magnitude from c.fcc (1-by-np)
10 %    c.diffmm  = First difference of code c.mm (1-by-np)
11 %    c.x0y0     = Coordinates where the code starts (1-by-2)
12 %
13 % C = FCHCODE(B, CONN) produces the same outputs as above, but
14 % with the code connectivity specified in CONN. CONN can be 8 for
15 % an 8-connected chain code, or CONN can be 4 for a 4-connected
16 % chain code. Specifying CONN=4 is valid only if the input
17 % sequence, B, contains transitions with values 0, 2, 4, and 6,
18 % exclusively.
19 %
20 % C = FHCODE(B, CONN, DIR) produces the same outputs as above, but,
21 % in addition, the desired code direction is specified. Values for
22 % DIR can be:
23 %
24 % 'same'      Same as the order of the sequence of points in b.
25 %               This is the default.
26 %
27 % 'reverse'   Outputs the code in the direction opposite to the
28 %               direction of the points in B. The starting point
29 %               for each DIR is the same.
30 %
31 % The elements of B are assumed to correspond to a 1-pixel-thick,
32 % fully-connected, closed boundary. B cannot contain duplicate
33 % coordinate pairs, except in the first and last positions, which
34 % is a common feature of boundary tracing programs.
35 %
36 % FREEMAN CHAIN CODE REPRESENTATION
37 % The table on the left shows the 8-connected Freeman chain codes
38 % corresponding to allowed deltax, deltay pairs. An 8-chain is
39 % converted to a 4-chain if (1) if conn = 4; and (2) only

```

```

40 % transitions 0, 2, 4, and 6 occur in the 8-code. Note that
41 % dividing 0, 2, 4, and 6 by 2 produce the 4-code.
42 %
43 % -----
44 %      deltax | deltay | 8-code   corresp 4-code
45 % -----
46 %      0       1       0           0
47 %     -1       1       1
48 %     -1       0       2           1
49 %     -1      -1       3
50 %      0      -1       4           2
51 %      1      -1       5
52 %      1       0       6           3
53 %      1       1       7
54 %
55 %
56 % The formula z = 4*(deltax + 2) + (deltay + 2) gives the following
57 % sequence corresponding to rows 1-8 in the preceding table: z =
58 % 11,7,6,5,9,13,14,15. These values can be used as indices into the
59 % table, improving the speed of computing the chain code. The
60 % preceding formula is not unique, but it is based on the smallest
61 % integers (4 and 2) that are powers of 2.
62
63 % Copyright 2002-2004 R. C. Gonzalez, R. E. Woods, & S. L. Eddins
64 % Digital Image Processing Using MATLAB, Prentice-Hall, 2004
65 % $Revision: 1.6 $ $Date: 2003/11/21 14:34:49 $
66
67 % Preliminaries.
68 if nargin == 1
69     dir = 'same';
70     conn = 8;
71 elseif nargin == 2
72     dir = 'same';
73 elseif nargin == 3
74     % Nothing to do here.
75 else
76     error('Incorrect number of inputs.')
77 end
78 [np, nc] = size(b);
79 if np < nc
80     error('B must be of size np-by-2.');
81 end
82
83 % Some boundary tracing programs, such as boundaries.m, output a
84 % sequence in which the coordinates of the first and last points are
85 % the same. If this is the case, eliminate the last point.
86 if isequal(b(1, :), b(np, :))
87     np = np - 1;

```

```

88     b = b(1:np, :);
89 end
90
91 % Build the code table using the single indices from the formula
92 % for z given above:
93 C(11)=0; C(7)=1; C(6)=2; C(5)=3; C(9)=4;
94 C(13)=5; C(14)=6; C(15)=7;
95
96 % End of Preliminaries.
97
98 % Begin processing.
99 x0 = b(1, 1);
100 y0 = b(1, 2);
101 c.x0y0 = [x0, y0];
102
103 % Make sure the coordinates are organized sequentially:
104 % Get the deltax and deltay between successive points in b. The
105 % last row of a is the first row of b.
106 a = circshift(b, [-1, 0]);
107
108 % DEL = a - b is an nr-by-2 matrix in which the rows contain the
109 % deltax and deltay between successive points in b. The two
110 % components in the kth row of matrix DEL are deltax and deltay
111 % between point (xk, yk) and (xk+1, yk+1). The last row of DEL
112 % contains the deltax and deltay between (xnr, ynr) and (x1, y1),
113 % (i.e., between the last and first points in b).
114 DEL = a - b;
115
116 % If the abs value of either (or both) components of a pair
117 % (deltax, deltay) is greater than 1, then by definition the curve
118 % is broken (or the points are out of order), and the program
119 % terminates.
120 if any(abs(DEL(:, 1)) > 1) | any(abs(DEL(:, 2)) > 1);
121     error('The input curve is broken or points are out of order.')
122 end
123
124 % Create a single index vector using the formula described above.
125 z = 4*(DEL(:, 1) + 2) + (DEL(:, 2) + 2);
126
127 % Use the index to map into the table. The following are
128 % the Freeman 8-chain codes, organized in a 1-by-np array.
129 fcc = C(z);
130
131 % Check if direction of code sequence needs to be reversed.
132 if strcmp(dir, 'reverse')
133     fcc = coderev(fcc); % See below for function coderev.
134 end
135

```

```

136 % If 4-connectivity is specified, check that all components
137 % of fcc are 0, 2, 4, or 6.
138 if conn == 4
139     val = find(fcc == 1 | fcc == 3 | fcc == 5 | fcc == 7 );
140     if isempty(val)
141         fcc = fcc./2;
142     else
143         warning('The specified 4-connected code cannot be satisfied.')
144     end
145 end
146
147 % Freeman chain code for structure output.
148 c.fcc = fcc;
149
150 % Obtain the first difference of fcc.
151 c.diff = codediff(fcc,conn); % See below for function codediff.
152
153 % Obtain code of the integer of minimum magnitude.
154 c.mm = minmag(fcc); % See below for function minmag.
155
156 % Obtain the first difference of fcc
157 c.diffmm = codediff(c.mm, conn);
158
159 %-----%
160 function cr = coderev(fcc)
161 % Traverses the sequence of 8-connected Freeman chain code fcc in
162 % the opposite direction, changing the values of each code
163 % segment. The starting point is not changed. fcc is a 1-by-np
164 % array.
165
166 % Flip the array left to right. This redefines the starting point
167 % as the last point and reverses the order of "travel" through the
168 % code.
169 cr = fliplr(fcc);
170
171 % Next, obtain the new code values by traversing the code in the
172 % opposite direction. (0 becomes 4, 1 becomes 5, ... , 5 becomes 1,
173 % 6 becomes 2, and 7 becomes 3).
174 ind1 = find(0 <= cr & cr <= 3);
175 ind2 = find(4 <= cr & cr <= 7);
176 cr(ind1) = cr(ind1) + 4;
177 cr(ind2) = cr(ind2) - 4;
178
179 %-----%
180 function z = minmag(c)
181 %MINMAG Finds the integer of minimum magnitude in a chain code.
182 % Z = MINMAG(C) finds the integer of minimum magnitude in a given
183 % 4- or 8-connected Freeman chain code, C. The code is assumed to

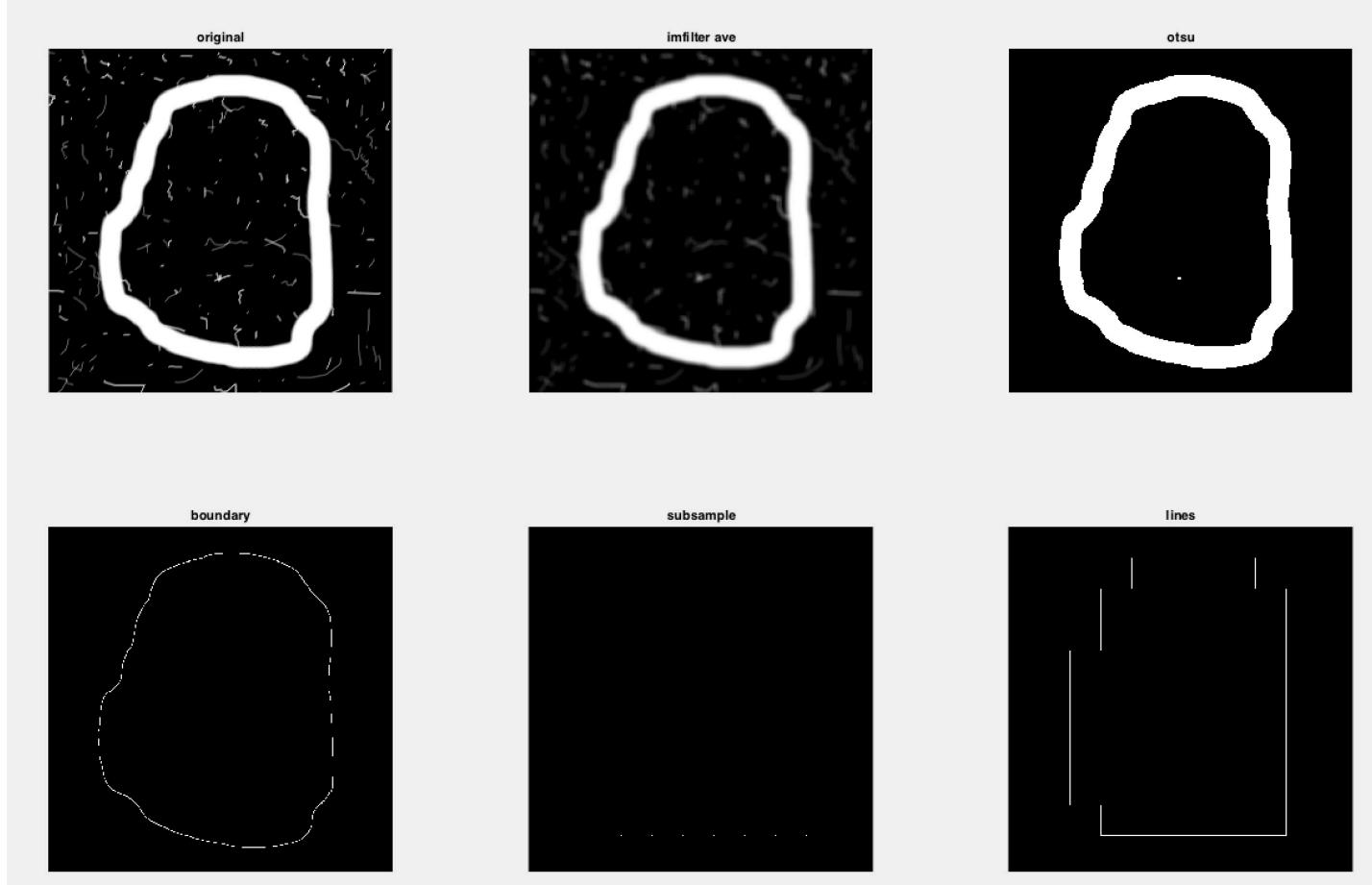
```

```

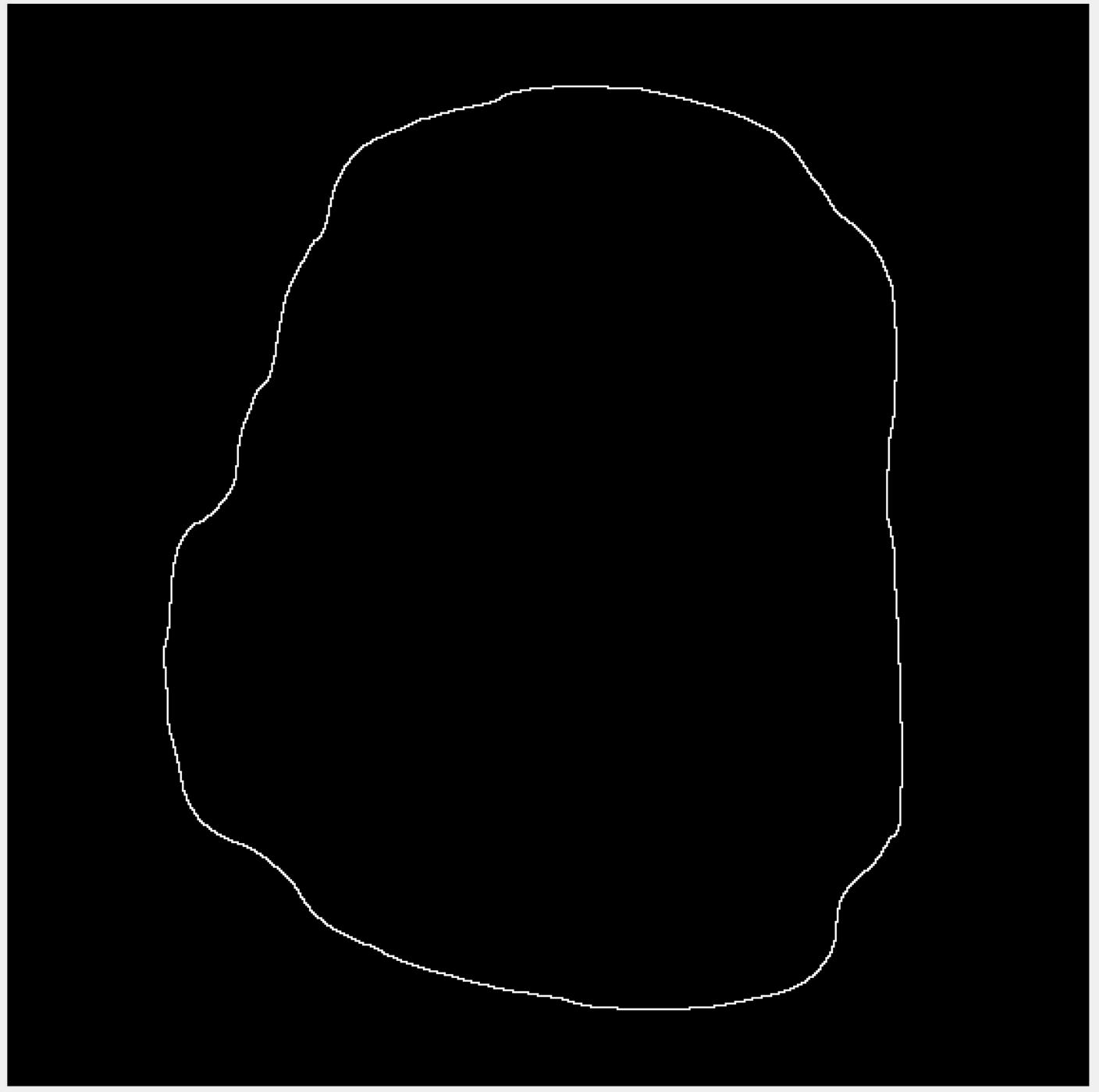
184 % be a 1-by-np array.
185
186 % The integer of minimum magnitude starts with min(c), but there
187 % may be more than one such value. Find them all,
188 I = find(c == min(c));
189 % and shift each one left so that it starts with min(c).
190 J = 0;
191 A = zeros(length(I), length(c));
192 for k = I;
193     J = J + 1;
194     A(J, :) = circshift(c,[0 -(k-1)]);
195 end
196
197 % Matrix A contains all the possible candidates for the integer of
198 % minimum magnitude. Starting with the 2nd column, successively find
199 % the minima in each column of A. The number of candidates decreases
200 % as the search moves to the right on A. This is reflected in the
201 % elements of J. When length(J)=1, one candidate remains. This is
202 % the integer of minimum magnitude.
203 [M, N] = size(A);
204 J = (1:M)';
205 for k = 2:N
206     D(1:M, 1) = Inf;
207     D(J, 1) = A(J, k);
208     amin = min(A(J, k));
209     J = find(D(:, 1) == amin);
210     if length(J)==1
211         z = A(J, :);
212         return
213     end
214 end
215
216 %-----%
217 function d = codediff(fcc, conn)
218 %CODEDIFF Computes the first difference of a chain code.
219 % D = CODEDIFF(FCC) computes the first difference of code, FCC. The
220 % code FCC is treated as a circular sequence, so the last element
221 % of D is the difference between the last and first elements of
222 % FCC. The input code is a 1-by-np vector.
223 %
224 % The first difference is found by counting the number of direction
225 % changes (in a counter-clockwise direction) that separate two
226 % adjacent elements of the code.
227
228 sr = circshift(fcc, [0, -1]); % Shift input left by 1 location.
229 delta = sr - fcc;
230 d = delta;
231 I = find(delta < 0);

```

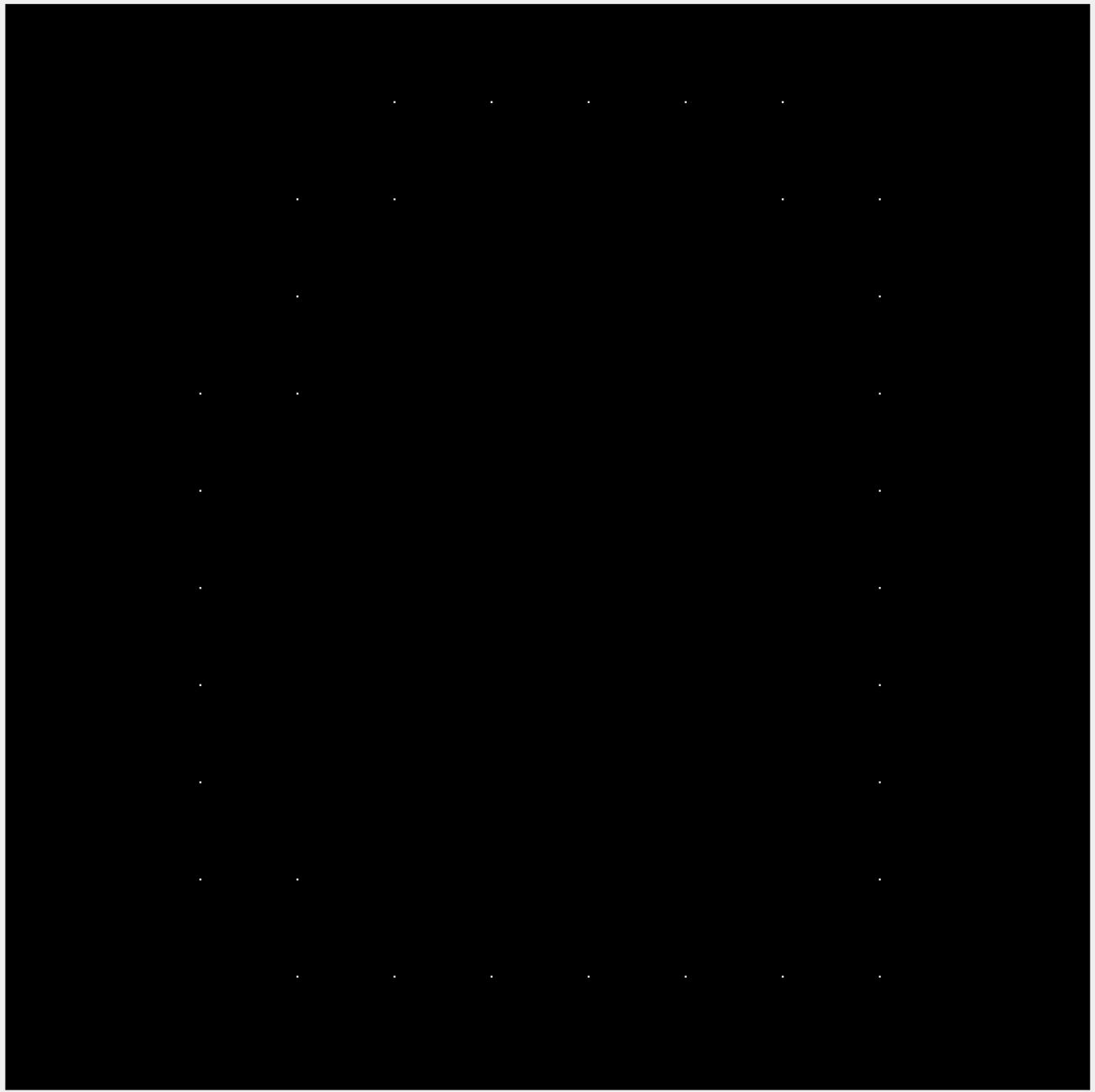
```
232  
233 type = conn;  
234 switch type  
235 case 4 % Code is 4-connected  
236 d(I) = d(I) + 4;  
237 case 8 % Code is 8-connected  
238 d(I) = d(I) + 8;  
239 end
```



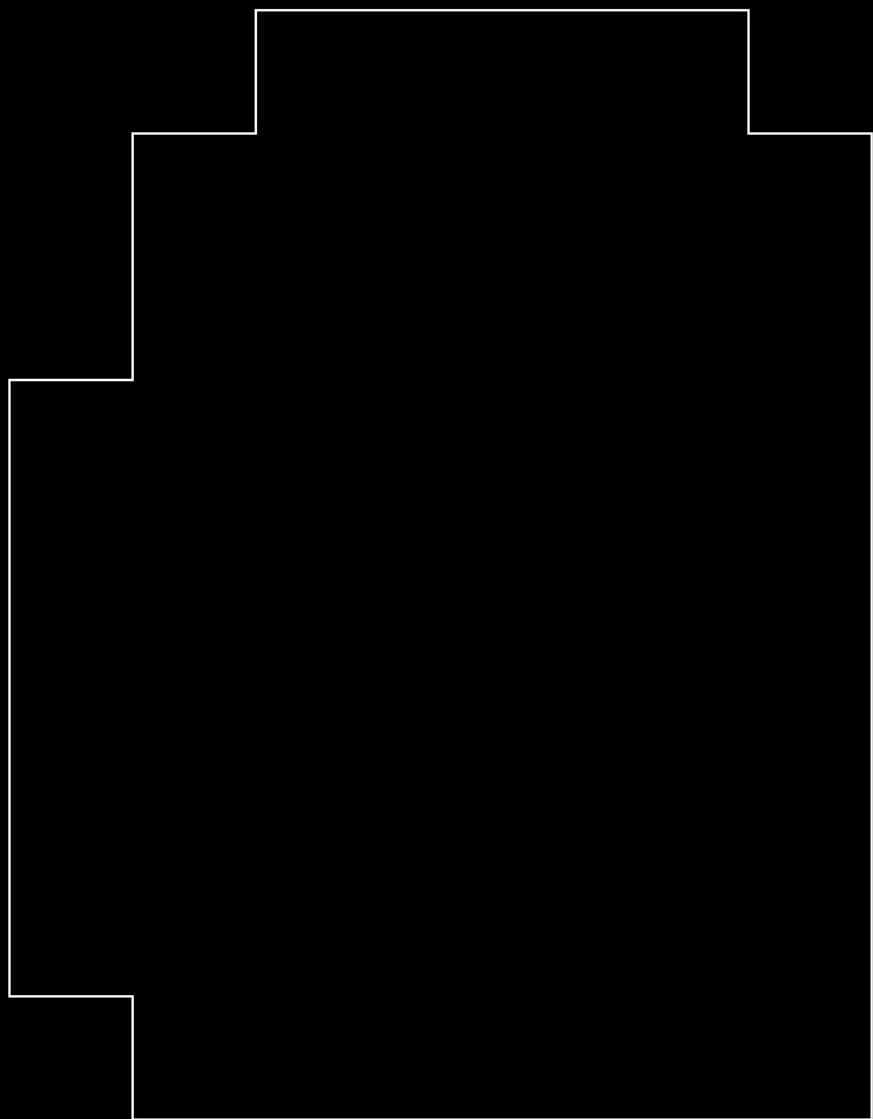
boundary image



subsample image



connected lines image



chain code result

MATLAB

```
1 >> cn_result.fcc
2
3 ans =
4
5 Columns 1 through 15
6
7     4     6     4     6     6     4     6     6     6     6     6     0     6     0
8
9 Columns 16 through 30
```



```

58      2     0     0     0     2     6     2     0     6     2     0     0     0     0
59
60  Columns 31 through 32
61
62      6     2
63
64
65 >>

```

Principal Components

pc.m

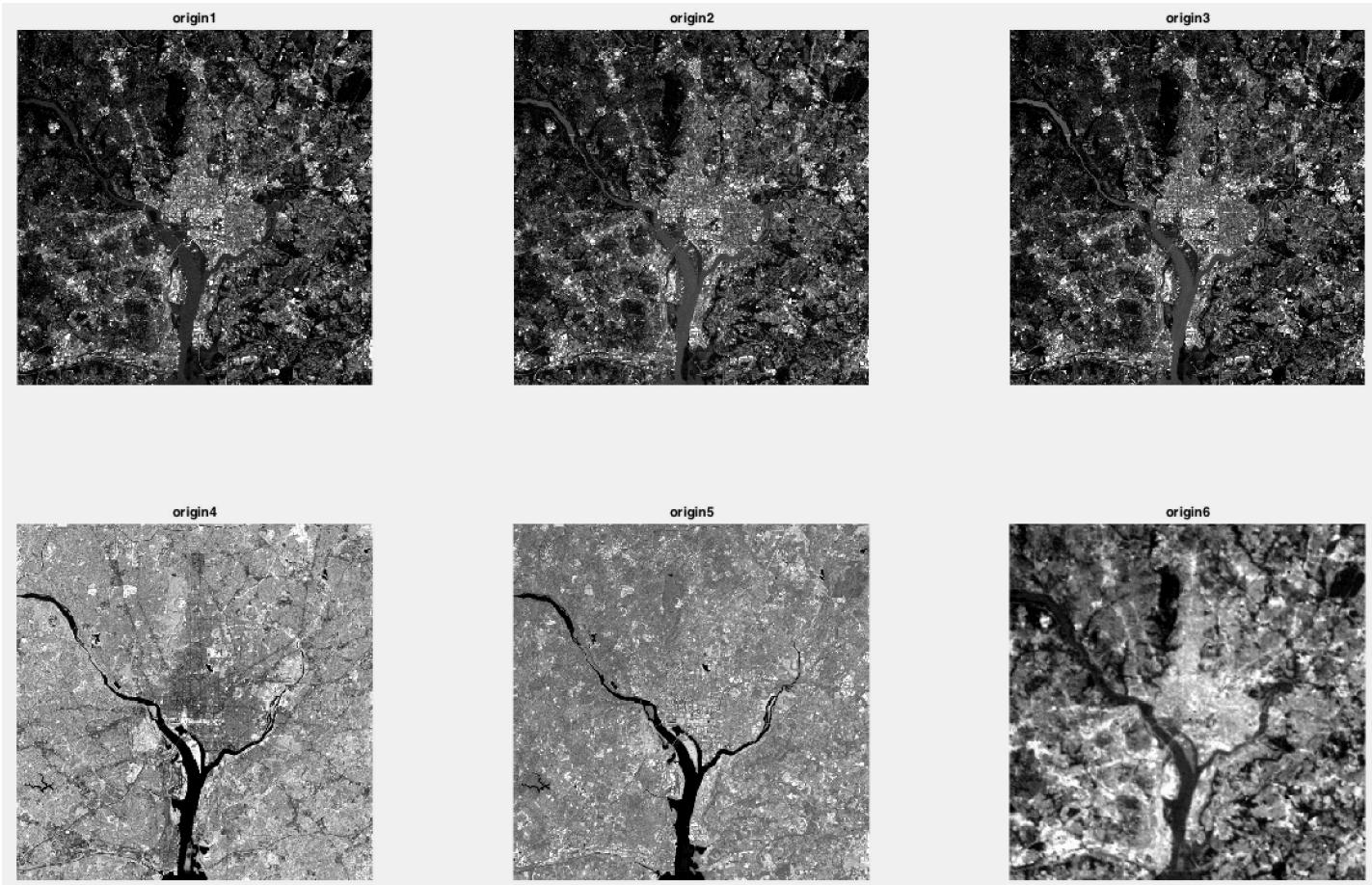
MATLAB

```

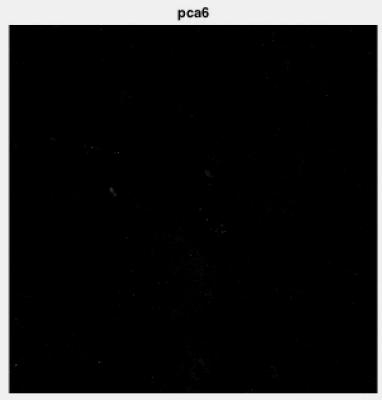
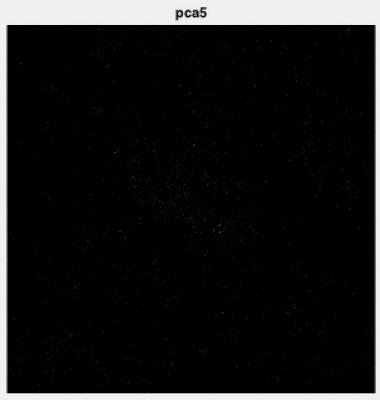
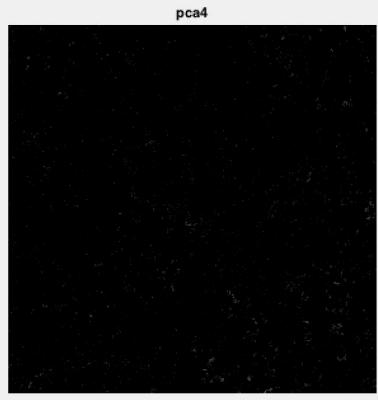
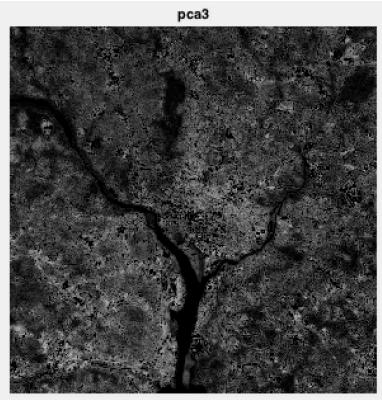
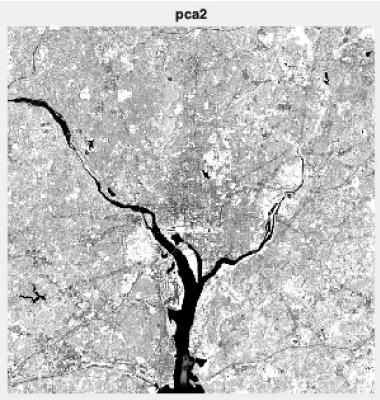
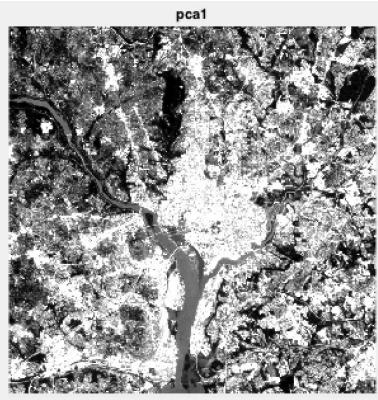
1 clear;clc;
2 path = '/Users/Personals/Desktop/DIP/assignments/prob10/washingtonDC/';
3 comb_img = [];
4
5 for i=1:6
6     wdc = double(imread([path 'WashingtonDC_Band' num2str(i) '.tif']));
7     wdc_col = reshape(wdc,564*564,1);
8     comb_img = [comb_img wdc_col];
9 end
10
11 comb_img = double(comb_img);
12 mu = mean(comb_img,2);
13 [coeff,score,latent] = pca(comb_img);
14 wdc_pcs = comb_img * coeff;
15 pca_result = zeros(564,564,6);
16 wdc_rec = bsxfun(@plus,score(:,1:2) * coeff(:,1:2)',mu);
17 rec_result = zeros(size(pca_result));
18 diff_result = zeros(size(pca_result));
19 comb_result = zeros(size(pca_result));
20
21 for i=1:6
22     comb_result(:,:,i) = reshape(comb_img(:,:,i),564,564);
23     pca_result(:,:,i) = reshape(wdc_pcs(:,:,i),564,564);
24     rec_result(:,:,i) = reshape(wdc_rec(:,:,i),564,564);
25     diff_result(:,:,i) = imsubtract(comb_result(:,:,i),rec_result(:,:,i));
26
27     subplot(2,3,i),imshow(uint8(comb_result(:,:,i)));title(['origin' num2str(i)]);
28     subplot(2,3,i),imshow(uint8(pca_result(:,:,i)));title(['pca' num2str(i)]);
29     subplot(2,3,i),imshow(uint8(rec_result(:,:,i)));title(['rec' num2str(i)]);
30     subplot(2,3,i),imshow(uint8(diff_result(:,:,i)));title(['diff' num2str(i)]);
31 end

```

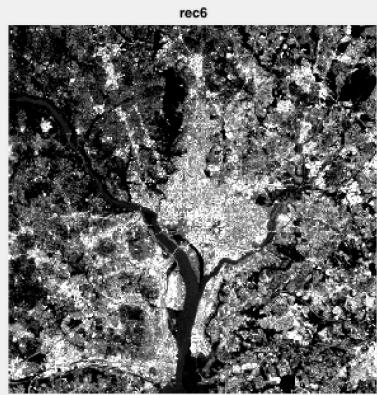
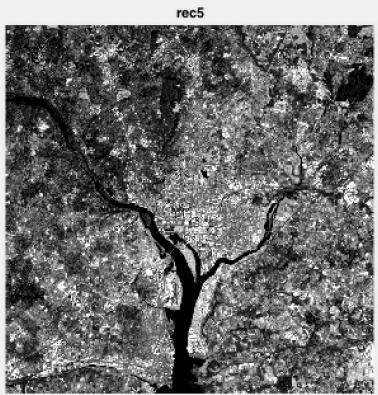
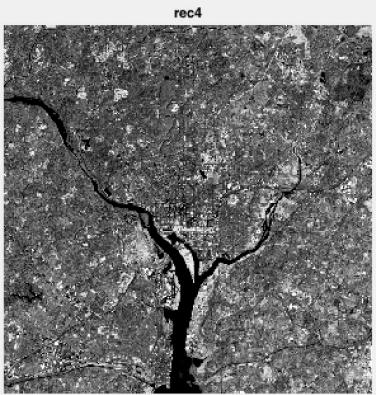
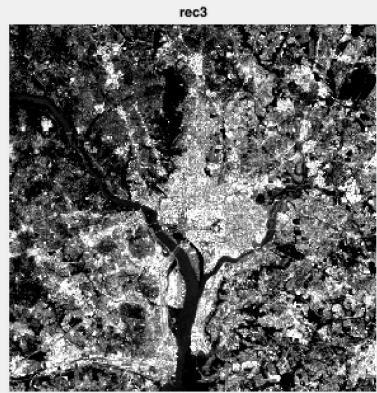
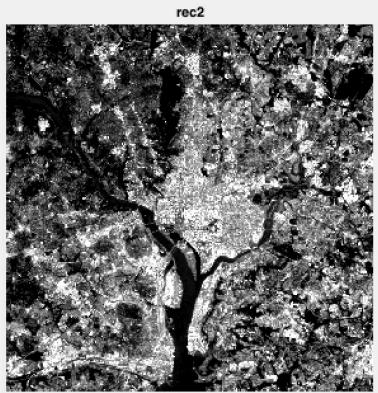
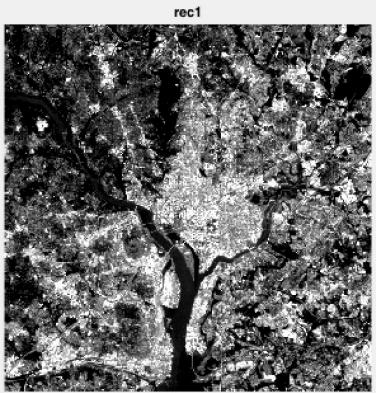
original image



principal components

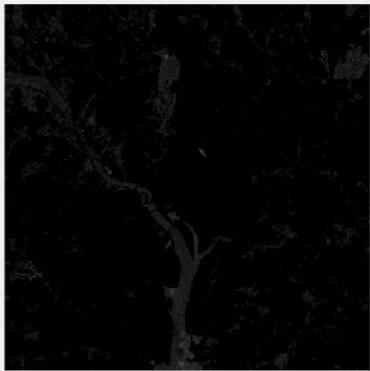


reconstruct image

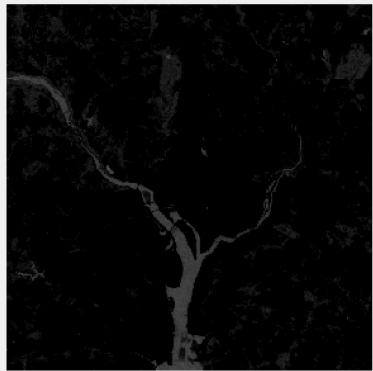


difference image

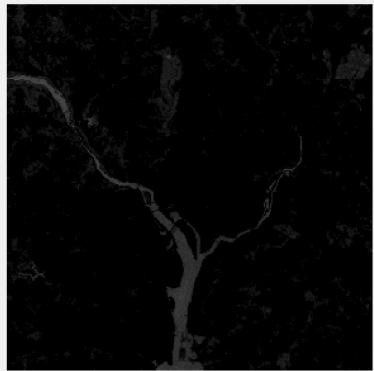
diff1



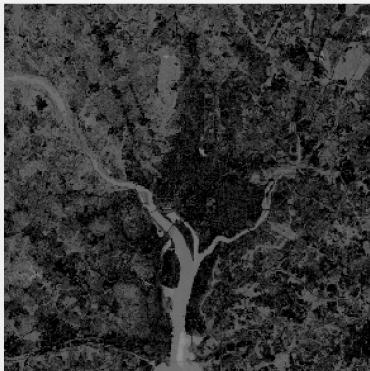
diff2



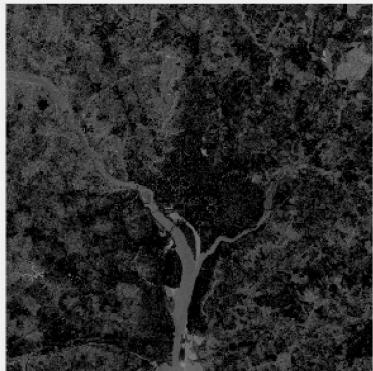
diff3



diff4



diff5



diff6

