# Skeletonization for 2D binary data by curvature.

```
program Skeletonization by curvature;
INPUT : x_size × y_size binary data. (* data format : PGM file *)
OUTPUT : " OUTNAME***.pgm " (* skeleton data. *)
         " OUTNAME_curve***.pgm " (* boundary data with curvature code. *)
         " error.txt " (* error coordinate. *)

type
    STRUCTURE = record
        sign : integer;
        x_coordinate : integer;
        y_coordinate : integer;
    end;

var curve_sequence : array["number of boundary pixels"] of STRUCTURE;
var i,j,k,hole_num:integer;
var hole_x,hole_y:array[1.."number of holes"] of integer;
(* hole_num: number of holes,hole_x,hole_y:coordinate of each holes *)
(* It is nessesary to set these hole datas before process. *)
var data,move,leave,watch,data_bin:array[1..x_size][1..y_size] of unsigned char;
(* data:Input data, move:moved data, leave:left data , watch:reference ,data_bin:binarized Input data*)
var curv_data:array[1..x_size][1..y_size] of integer;
(* curv_data:for curvature *)

begin
    data[x][y] := "Input 2D Data";
    leave_pix(data,leave);
    while "there are one-pixel thickness components." do
        begin
            cut(data);
        end;
    for i := 1 to x_size do
        for j := 1 to y_size do
            data_bin[i][j] := "binarized data[i][j]";
    curv_cul(data_bin,curv_data);
    curv_num(data_bin,curv_data);
    if "there are holes." then
        for k := 1 to hole_num do
            for i := hole_x[k] to x_size do
                for j := hole_y[k] to y_size do
```

```
28:                 if data_bin[i][j] = BOUNDARY then
29:                     begin
30:                         bound_rev(data,watch,curve_sequence,i,j);
31:                         moving(move,curve_sesquence);
32:                     end
33:     else
34:         while "there are boundarys."do
35:             begin
36:                 for i := 0 to x_size do
37:                     for j := 0 to y_size do
38:                         if data_bin[i][j] = BOUNDARY then
39:                             begin
40:                                 bound(data,watch,curve_sequence,i,j);
41:                                 moving(move,curve_sequence);
42:                             end
43:             end
44:     paint(move);
45:     "reverse the processed image.";(* IMAGE → BACK, BACK → IMAGE *)
46:     if "there are components more than one-pixel thickness" then
47:         "Iterate from the line 13.";
48:     for i := 1 to x_size do
49:         for j := 1 to y_size do
50:             if (move[i][j][k] = Image or leave[i][j][k] = Image) then
51:                 "Output Skeleton Data" := Image;
52: end.
```

```
(* leave component composed one-pixel thickness *)
1: procedure leave-pix(var data,leave:array[x_size][y_size] of unsigned char);
2: begin
3:     for i := 1 to x-size do
4:         for j := 1 to y-size do
5:                 if data[i][j] = one-pixel thickness then
6:                     leave[i][j] = LEAVE;
7: end;
```

```
(* eliminate pixel connected by one pixel *)
1: procedure cut(var data:array[x_size][y_size] of unsigned char);
2: begin
3:     for i := 1 to x-size do
4:         for j := 1 to y-size do
5:                 if 'data[i][j] = one-pixel thickness' then
6:                     data[i][j] = BACK;
```

```
end;
```

```
(* calculate curvature for edge pixel *)
procedure curv_cul(var data_bin:array[x_size][y_size] of unsigned char , curv_data:array[x_size][y_size]
of integer);
begin
    for i := 1 to x-size do
        for j := 1 to y-size do
            'calculate curvature for data[i][j]';
end;
```

```
(* assign curvature code *)
procedure curv_num(var data_bin:array[x_size][y_size] of unsigned char , curv_data:array[x_size][y_size]
of integer);
begin
    for i := 1 to x-size do
        for j := 1 to y-size do
            if curv_data[i][j] > 0 then
                data_bim[i][j] := PLUS;
            else if curv_data[i][j] < 0 then
                data_bin[i][j] := MINUS;
            else data_bin[i][j] := ZERO;
end;
```

```
(* boundary tracking (clockwise) for outer boundary *)
procedure bound(var data,watch:array[x_size][y_size] of unsigned char , curv_sequence:array["number
of boundary pixels"] of curve_sequence , x_start,y_start:integer);
var x_now,y_now:integer;
begin
    while x_now <> x_start and y_now <> y_start do
        begin
            'curve_sequence.sign := curvature code of data[x_now][y_now];
            'curve_sequence.x_coordinate := x_now;
            'curve_sequence.y_coordinate := y_now;
            'watch[x_now][y_now] := BACK';
             (* for reference. if watch is null then all boundarys are checked. *)
             " boundary tracking by 4-neighborhood ";
         end
end;
```

```
(* boundary tracking (counterclockwise) for inner boundary *)
procedure bound_rev(var data,watch:array[x_size][y_size] of unsigned char , curv_sequence:array["number
```

```
of boundary pixels"] of curve_sequence , x_start,y_start:integer);
var x_now,y_now:integer;
begin
    while x_now <> x_start and y_now <> y_start do
        begin
            'curve_sequence.sign := curvature code of data[x_now][y_now];
            'curve_sequence.x_coordinate := x_now;
            'curve_sequence.y_coordinate := y_now;
            'watch[x_now][y_now] := BACK';
        (* for reference. if watch is null then all boundarys are checked. *)
            " boundary tracking by 4-neighborhood ";
        end
end;
```

```
(* move pixel according to curvature *)
procedure moving(var move:array[x_size][y_size] of unsigned char , curv_sequence:array["number
of boundary pixels"] of curve_sequence , h:integer(* number of boundary pixels*));
var x_now,y_now:integer;
begin
    "move boundary pixels according to curvature";
end;
```

```
(* fill boundary *)
procedure paint(var move:array[x_size][y_size]);
begin
    "fill background";(* then reverse image *)
end;
```