Technical University of Cluj - Napoca
Computer Science Department

Image Processing

(Year III, 2-nd semester)

Binary Images: Object Labeling. Contour Tracing (III)

Binary Algorithms - Definitions

1. Neighbors

-two pixels are **4-neighbors** or **d-neighbors** if they share a common side

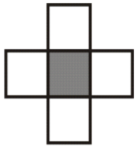
-two pixels are **i-neighbors** if they share a common corner

-the **neighbour** concept includes both **d-neighbour** and **i-neighbour**

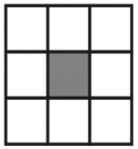
-two pixels are neighbors or **8-neighbors** if they share at least a corner

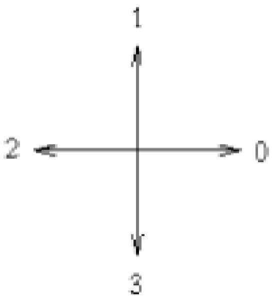
-the N-neighbour is the neighbour in direction N, where N is a direction code and $0 \le N \le 3$ or $0 \le N \le 7$

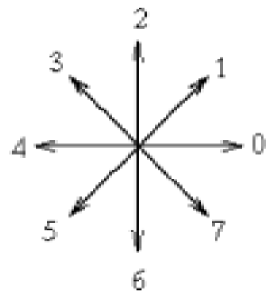
N4



N8







A pixel is said to be 4-connected to its 4-neighbors

and 8-connected to its 8-neighbors.

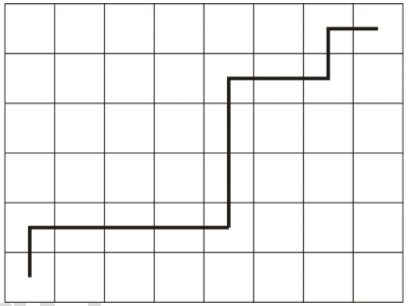
2. Path

Path $(p[i_0, j_0] \Rightarrow p[i_n, j_n]) := \{[i_0, j_0], [i_1, j_1], \ldots, [i_n, j_n] \mid [i_k, j_k]\ N_{4/8}\ [i_{k+1}, j_{k+1}]\ \forall\ k = 0 \ldots n-1\}$

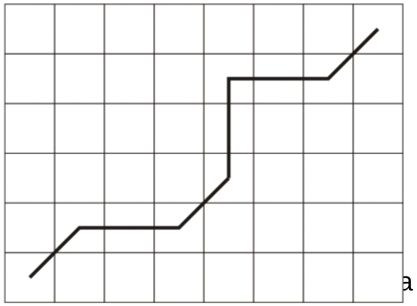
N4 $\Rightarrow$ 4-path or d-path

N8 $\Rightarrow$ 8-path or i-path

4-path



8-path



Binary Algorithms - Definitions

3. Foreground $S := \{ p[i,j] \mid p[i,j] = 1 \}$

4. Connectivity $p \leftrightarrow q$ (connected) if $\exists$ Path $(p \Rightarrow q) \subset S$, $p \in S$, $q \in S$.

5. Connected points $\{p_i \in S, i = 1 \ldots n \mid p_k \leftrightarrow p_j, \forall (p_k, p_j) \in S, k,j = 1 \ldots n\}$

6. Boundary Boundary (S): = $S' = \{ p \in S \mid \exists q \in N_{4/8}(p), q \in C(S) \}$

C(S) – complement of S

7. Interior Interior (S) = S - S'

8.Connected component

Maximal set of connected points $\{p_i \in S, i = 1 \ldots n \mid p_k \leftrightarrow pj, \forall (p_k, p_j) \in S, k,j = 1 \ldots n\}$

9. Background set of all connected components belonging to C(S) that have points on the border of an image. All other components of the image belonging to C(S) are called holes.

Technical University of Cluj Napoca
Computer Science Department

Labeling Connected Components

Connected component : maximal set of connected points

$\{p_i \in S , i = 1 \ldots n \mid p_k \leftrightarrow pj, \forall (p_k, p_j) \in S, \ k,j = 1 \ldots n\}$

One way to label the objects in a discrete binary image is to choose a point where b_{ij} =1 and assign a label to this point and its neighbors. Next, label all the neighbors of these neighbors, and so on.

- When this recursive procedure terminates, one component will have been labeled completely, and we can continue by choosing another start point.
- To find a new start point, we can simply scan through the image in a systematic way, starting a labeling operation whenever an unlabeled point is found where b_{ij} =1.

$\Rightarrow$
Labeling

Sequential Labeling

Iterative Algorithm (Haralick 1981)

- no auxiliary storage to produce the labeld image from the binary image.
- useful in environments whose storage is severely limited.

1. initialization step
2. repeat

        top-down & left-right label propagation

        bottom-up & right-left label propagation

    until no changes occur

```
procedure Iterate;
"Initialization of each 1-pixel to a unique label"
for L:=1 to NLINES do
        for P:=1 to NCOLUMNS do
                if  I(L,P) =1
                then LABEL(L,P):=NEWLABEL()
                else LABEL(L,P):=0
        end for
end for;
```

Sequential Labeling

a	b	c
d	e	

	e	d
c	b	a

	e	

"Top-down passes; Bottom up passes; 8-connected neighborhood"

"**procedure** Iterate – page 2"

"Iteration of top-down followed by bottom-up passes"

```
repeat
"Top-down passes"
CHANGE:=false;
 for L:=1 to NLINES do
        for P:=1 to NCOLUMNS do
                if  LABEL(L,P)<>0 then
                        begin
                                M:=MIN(LABELS(NEIGHBORS(L,P)U(L,P)));
                                if M<> LABEL(L,P)
                                then CHANGE:=true;
                                LABEL(L,P):=M
                        end
        end for
 end for;
```

Sequential Labeling

“**procedure** Iterate – page 3”

“Bottom-up pass”

```
 for L:= NLINES to 1 by –1 do
          for P:= NCOLUMNS to 1 by –1 do
                    if  LABEL(L,P)<>0 then
                              begin
                                        M:=MIN(LABELS(NEIGHBORS(L,P)U(L,P)));
                                        if M<> LABEL(L,P)
                                        then CHANGE:=true;
                                        LABEL(L,P):=M
                              end
          end for
end for;

until CHANGE:=false

end Iterate
```


Technical University of Cluj Napoca
Computer Science Department

Sequential Labeling

Example (N4)

	1	1		1	1	
	1	1		1	1	
	1	1	1	1	1	

1. Initial image

	1	2		3	4	
	5	6		7	8	
	9	10	11	12	13	

2. Initialization

	1	1		3	3	
	1	1		3	3	
	1	1	1	1	1	

3. Top-down & left-right label propagation

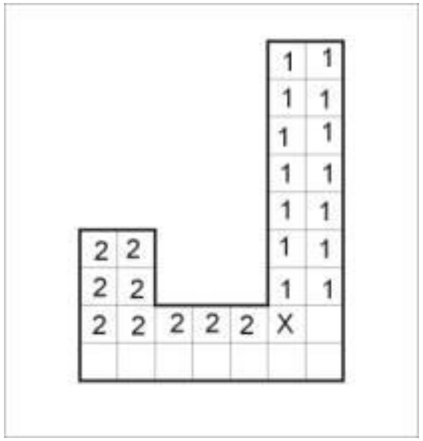
	1	1		1	1	
	1	1		1	1	
	1	1	1	1	1	

4. Bottom-up & right-left label propagation

Classical Algorithm

- Based on the classical connected components algorithm for graphs.
- 2 passes through the image but requires a large global table for recording equivalences.

1. First pass: performs label propagation, much as described above.
   - Whenever a situation arises in which two different labels can propagate to the same pixel, the smaller label propagates and each such equivalence found is entered in an equivalence table (e.g. (1,2) $\rightarrow$ EqTable).
   - Each entry in the equivalence table consists of an ordered pair, the values of its components being the labels found to be equivalent.
   - After the first pass, the equivalence classes are found.
   - Each equivalence class is assigned a unique label, usually the minimum (or oldest) label in the class.



2. A second pass through the image performs a translation, assigning to each pixel the label of the equivalence class of its 1-st pass label.

Classical Algorithm

```
procedure Classical
"Initialize global equivalence table"
EQTABLE:=CREATE();

"Top-down pass 1"
for L:= 1 to NLINES do
        "Initialize all labels on line L to zero"
        for P:= 1 to NCOLUMNS do
                LABEL(L,P):=0
        end for
        "Process the line"
        for P:=1 to NCOLUMNS do
                if I(L,P):= 1 then
                        begin
                                A:= NEIGHBORS((L,P));
                                if ISEMPTY(A)
                                then M:=NEWLABEL()
                                else M:= MIN(LABELS(A));
                                LABEL(L,P):=M;
                                for X in LABELS(A) and X<>M
                                        ADD(X, M, EQTABLE)
                                end for;
                        end
        end for
end for;
```

Classical Algorithm

```
"Find the equivalence classes"
EQCLASSES:=Resolve(EQTABLE);

"Find the equivalence label of an equivalence class"
for E in EQCLASSES
        EQLABEL(E):= min(LABELS(E))
end for;

"Top-down pass 2"
for L:= 1 to NLINES do
        for P:= 1 to NCOLUMNS do
                if I(L,P) = 1
                then LABEL(L,P):=EQLABEL(CLASS(LABEL(L,P)))
        end for
end for
end Classical
```

- RESOLVE - algorithm for finding the connected components of the graph structure, defined by the set of equivalences (EQTABLE) defined in pass 1.
- The main problem with the classical algorithm is the global equivalence table (large images with many regions, the equivalence table can become very large)

Technical University of Cluj Napoca
Computer Science Department

Classical Algorithm

Example (N4)

1					1	1
		1	1			1
		1				1
		1				1
1	1	1				1
	1	1		1		1
	1	1	1	1		1
				1	1	1

1. Initial image

1					2	2
		3	3			2
		3				2
		3				2
4	4	3				2
	4	3		5		2
	4	3	3	3		2
				3	3	2

2. Top down (pass 1)

EQTABLE:
(4, 3), (3, 5), (3, 2) …

EQCLASSES:
1: {4, 3, 5, 2}
2: (6,8,9, …}
….
n: {….}

EQLABEL:
1, 2
2, 6
…
n, x

Classical Algorithm (improvement)

A Space-Efficient Two-Pass Algorithm That Uses a Local Equivalence Table

⇒ use of a small local equivalence table that stores only the equivalences detected from the current and previous lines

Maximum number of equivalences = number of pixels / line.

1. First pass:

    the equivalences from one line are used in the propagation step to the next line.

1. Second pass is required:

    The new equivalence classes and labels finding followed by assigning the final labels.

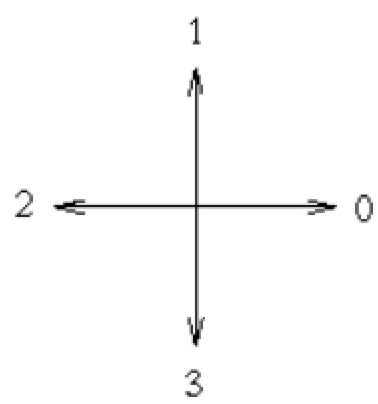
Contour Tracing

Boundary/contour

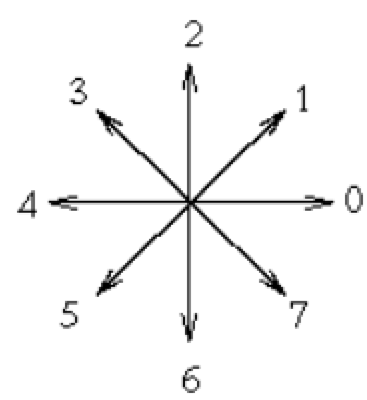
Contour(R) = $\{ p \in R \mid \exists q \in N4/8(p), q \in C(R) \}$

- The ***contour*** or ***i-contour* of R** (where R is a connected set of pixels) is defined as the set of all pixels in R which have at least one *d-neighbor* not in R.
- The ***d-contour* of R** is the set of all pixels in R, which have at least one neighbor not in R.
- *N-neighbor* (*chain-code / direction codes*): *c*

  (numerical operations on *c* are assumed to be modulo 4 or 8)



4-neighbour



8-neighbour

Contour Tracing

Single contour tracing

- The algorithm can be described in terms of an observer who walks counterclockwise along pixels belonging to the set and selects the rightmost pixel available.
- The tracing terminates when the current pixel is the same as the initial pixel.
- The TRACER algorithm must be applied once for each hole of a region, in addition to one application for the external contour.
- Therefore, it must be combined with a search algorithm for locating holes in the interior of the region.
- Description of the contours (output): { $A(x_0, y_0, c_0)$, $C_i(x_i,y_i,c_i)$, i=1 … n}

TRACER Procedure

Notations:

A:	the starting point of the contour of the set R (can be found in a number of ways, including a top-to-bottom, left-to-right scan);
C:	the current point whose neighborhood is examined;
S :	the search direction in the terms of the direction codes;
first:	is a flag that is true only when the tracing starts;
found:	is a flag that is true when a next point on the contour is found;

```
if ((A:=NEXT_START_ELEMENT())!=NULL)
  {Q:=CREATE_LIST();
   TRACER(A,Q);
  }
```

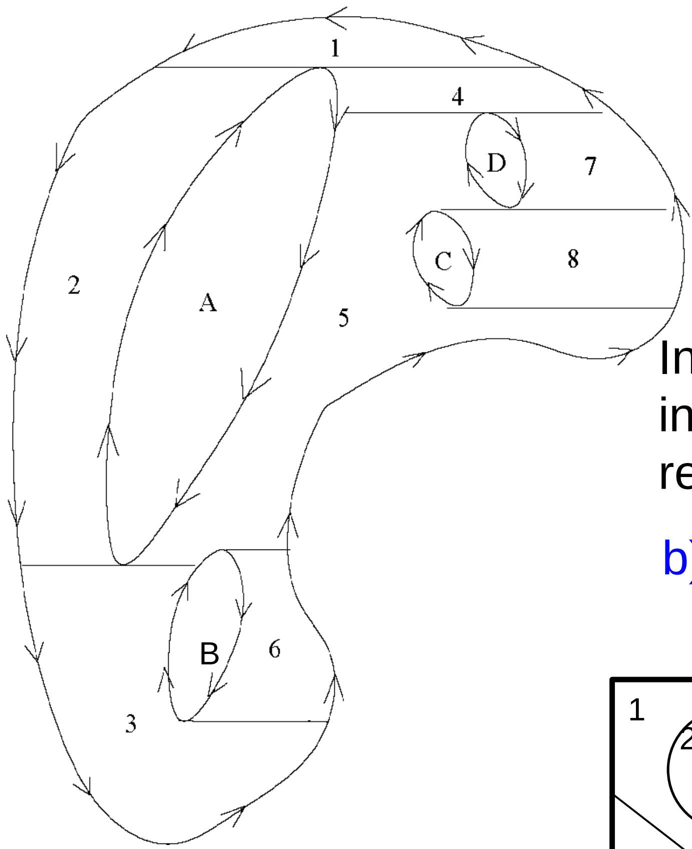
Contour Tracing

```
procedure TRACER(A,Q)
begin
        first:=TRUE;
        C:=A;
        S:=6:
        while((C!=A) or (first=TRUE))
                {
                found:=FALSE;
                count:=0;
                while((found=FALSE) and (count=<3))
                        {
                        if( B, the (S-1)-neighbor of C, is in R) then
                        { APPEND((C,S-1),Q),  C:=B,  S:=S-2,  found:=TRUE;}
                          else if (B, the S-neighbor of C, is in R) then
                             { APPEND((C,S),Q),  C:=B,  found:=TRUE;}
                                 else if (B, the (S+1)-neighbor of C, is in R) then
                                  {APPEND((C,S+1),Q),  C:=B,  found:=TRUE;}
                                  else
                                   {S:=S+2,  count:=count+1;}
                                 endif
                            endif
                        endif
                        }
                first:=FALSE;
                }
end
```

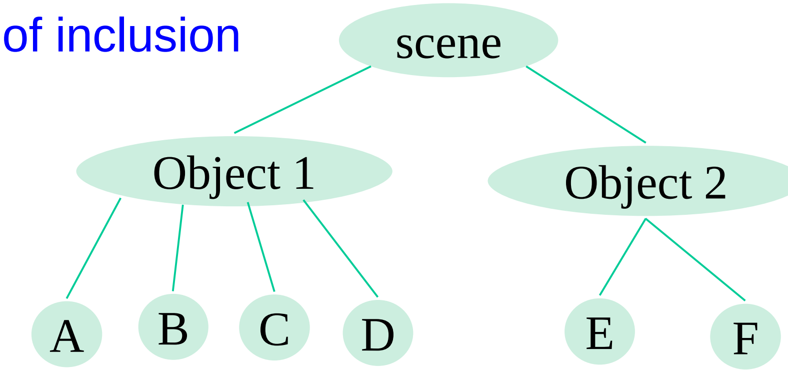

Technical University of Cluj Napoca
Computer Science Department

Traversal of all the contours of an image

Structural description of the scene:

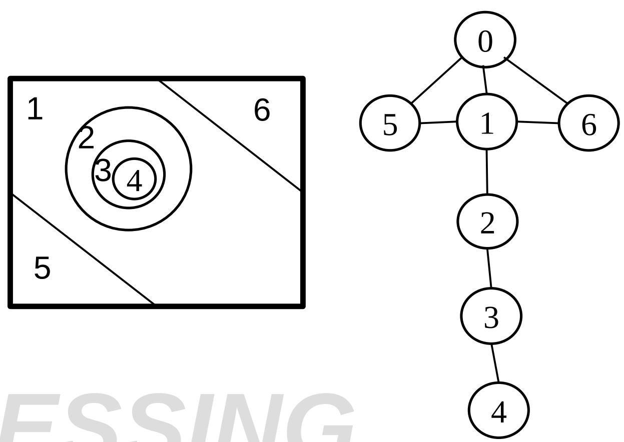


a) Tree of inclusion



In this tree the nodes represent the connected regions in the image and the edges represent the inclusion relations. The root models the whole scene.

b) Adjacency graph



The nodes represent the connected regions in the image and the edges link adjacent regions.

Traversal of all the contours of an image

Traversal of All the Contours of a Region

⇒ closed i-paths and follows external contours counterclockwise (TRACER()) and contours of hole clockwise (TRACER1()).

Changes:

- When a pixel is marked as the current point, its value is incremented by 1. Then, at the end of the tracing, pixels of the contour will have values 2 or greater. These values are used when the interior is searched for holes.
- After the external contour has been found and placed in the queue Q, we start examining the contents of the latter. If we find a point located on a downward arc, we start a search to the right. Such pixels can be characterized easily by the requirement that the previous element of the chain code must have values 4 to 7 while the next element should be in the range 5 to 7.
- While scanning along the horizontal direction one must search for either the start of a hole or the other side of the outside contour.
- The TRACER1() procedure is called when an unmarked start of a hole point is find.
- The extracted contours of the holes are inserted into the Q list.
- The content of the list is examined until its end.

Traversal of all the contours of an image

MULTI_TRACE Procedure

Notations:

P: current pixel with x, y coordinate;

c: direction code; value 8 is used to specify the beginning of a new contour;

c0: initial direction code;

Q: {(P, c)}

```
If ((A:=NEXT_START_ELEMENT())!=NULL)
        {
         Q:=CREATE_LIST();
         MULTI_TRACE(A,Q);
        }
```

Traversal of all the contours of an image

```
procedure MULTI_TRACE(A,Q)
begin
          TRACER(A,Q);
          while (Q !=NULL)
          {
                (P,c):=Remove(Q);
                if (c=8)
                    {
                       c0:=c;
                       (P,c):=Remove(Q);
                     }
                 if ((c0 ∈{4 : 7}) and (c∈{5 : 7}))
                     {
                        Starting from P search in the x-direction and examine triplets of
                        successive pixels, A, B, C.
                        if ((A!=0) and (B=1) and (C=0))
                             {
                                TRACER1(B,Q);
                                goto LABEL1 ;
                              }
                        If ((A=1) and (B=2) and (C=0))
                                goto LABEL1;
                      }
                LABEL1: c0=c;
          }
end
```

Polygonal Approximation

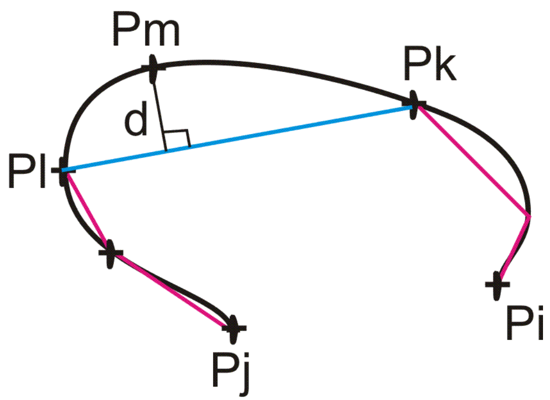
Polygonal approximation of contours

Curve C: $f(x,y)=0 \Rightarrow$ polygon that closely approximates C with an error smaller than ε and having a number of vertices as small as possible**:**

- Any polygonal fitting algorithm requires that the data points be subdivided into groups, each one of them to be approximated by a side of the polygon.
- The first simplification of the polygon fit problem is to draw a line between the endpoints of each group rather than search for the optimal solution.
- If the approximation error is too big the group could be split in two and so one until the error becomes acceptable.
- Let Q a contour consisting of P_i (x_i,y_i) where $i=1,2,\ldots,n$, and ε the error threshold.

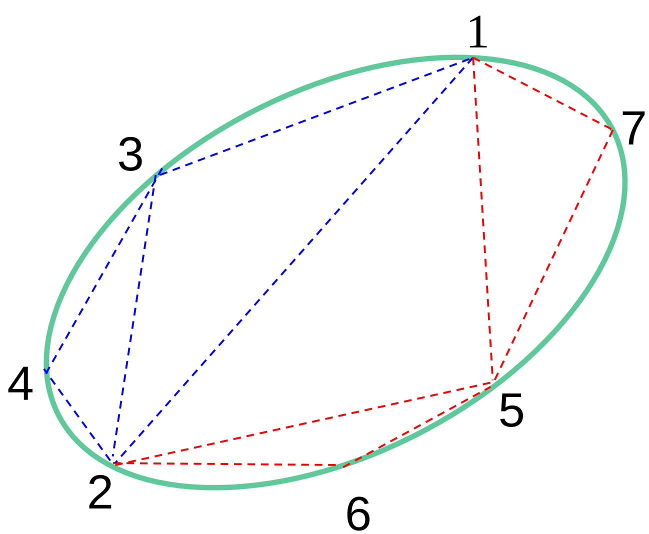
Polygonal Approximation

Procedure POLIGONAL_APROX(Q)
begin
A:=Create_List();
B:=Create_List();
i=Index_of_first_point(Q);
j=Index_of_the_most_far_point(Q);
Insert(j,A); Insert(j,B);
Insert(i,A);
while((A!=NULL)
{
Let *k* and *l* the indexes of the last elements of the lists A and B;
Let P_k P_l the segment generated by these two points;
Let *m* the index of the most far point to P_kP_l segment among the contour points starting with P_k and ending with P_l, when the contour is scanned in counterclock-wise direction.
if ((d=Distance($P_kP_{l,}$, P_m) > ε)
then Insert(m, A)
else { Delete(k, A)
Insert(k, B); }
}
end



$$\text{distance}(P_1, P_2, (x_0, y_0)) = \frac{|(y_2 - y_1)x_0 - (x_2 - x_1)y_0 + x_2y_1 - y_2x_1|}{\sqrt{(y_2 - y_1)^2 + (x_2 - x_1)^2}}.$$

Polygonal Approximation – Example



A	B
2 1 3 4	2
2 1 3	2 4
2 1	2 4 3
2	2 4 3 1
2 5	2 4 3 1

A	B
2 5 7	2 4 3 1
2 5	2 4 3 1 7
2	2 4 3 1 7 5
2 6	2 4 3 1 7 5

A	B
2	2 4 3 1 7 5 6
	2 4 3 1 7 5 6 2