Digital Image Processing

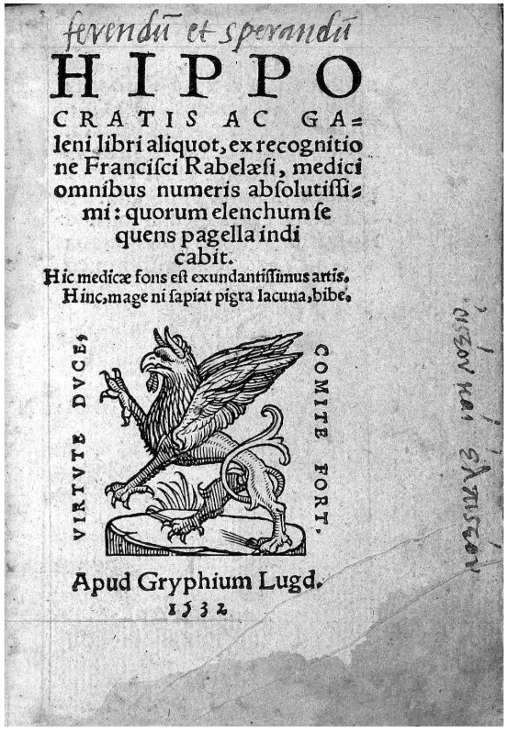
Lecture # 9 C
Morphological Image Processing

Run Length Smoothing Algorithm

Text/Graphics Segmentation

Text/Graphics Segmentation

Separate Text from Graphics



ferendū et sperandū

HIPPOCRATIS AC GA=
leni libri aliquot, ex recognitio
ne Franciſci Rabelæſi, medici
omnibus numeris abſolutiſſi=
mi: quorum elenchum ſe
quens pagella indi
cabit.

Hic medicæ fons eſt exundantiſſimus artis.
Hinc, mage ni ſapiat pigra lacuna, bibe.

VIRTVTE DVCE, COMITE FORT.

Apud Gryphium Lugd.
1532

[illegible]

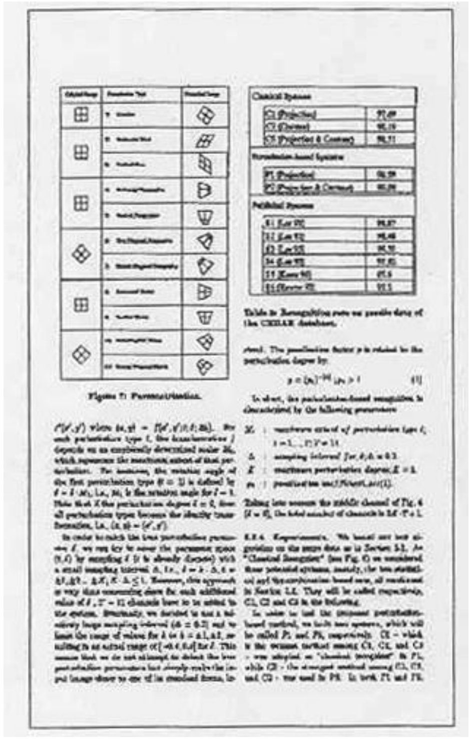


Figure 7: Parametrization.

[illegible]

Text/Graphics Segmentation

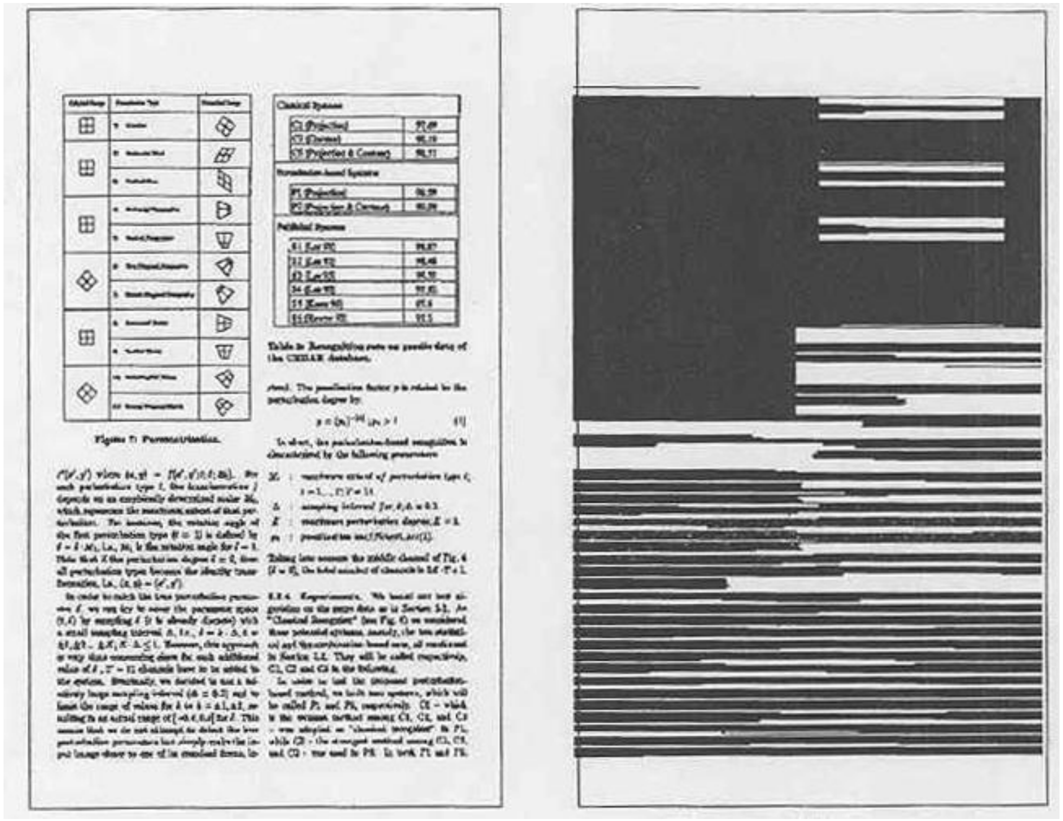
Run Length Smoothing Algorithm – RLSA

- Change runs of white pixels of length below a threshold to black
- Black pixels remain unchanged
- Example: C=4

X= 0 0 0 1 1 1 0 0 1 1 1 1 1 0 0 0 1 1 1 0

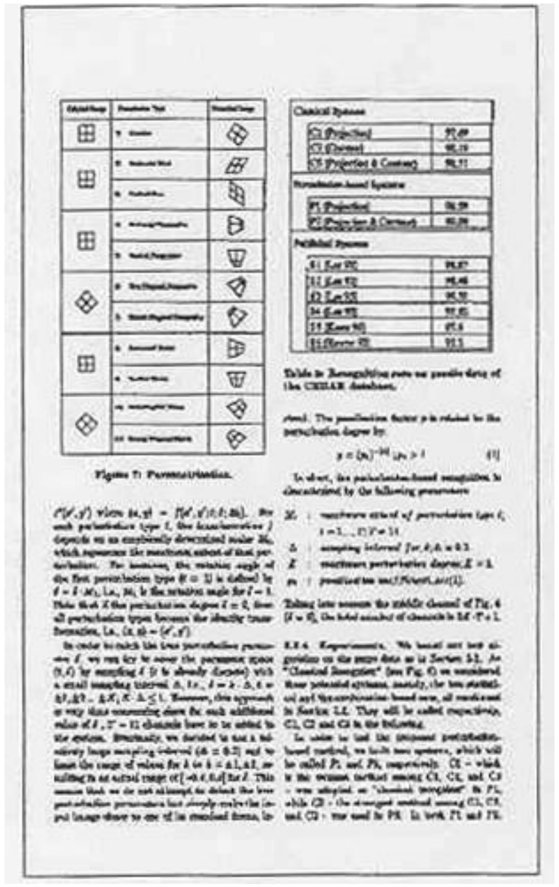
Y= 0 0 0 0 0 0 0 0 1 1 1 1 1 0 0 0 0 0 0 0

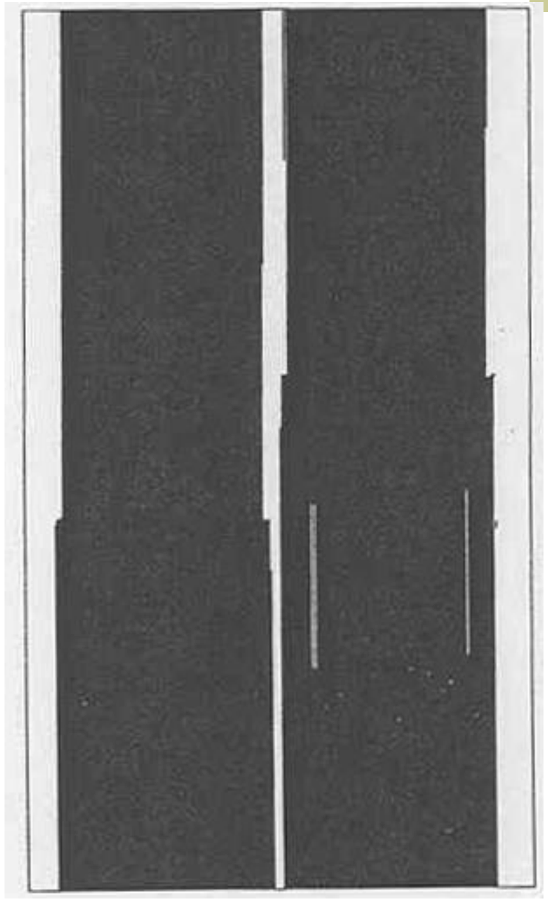
Text/Graphics Segmentation



Horizontal RLSA

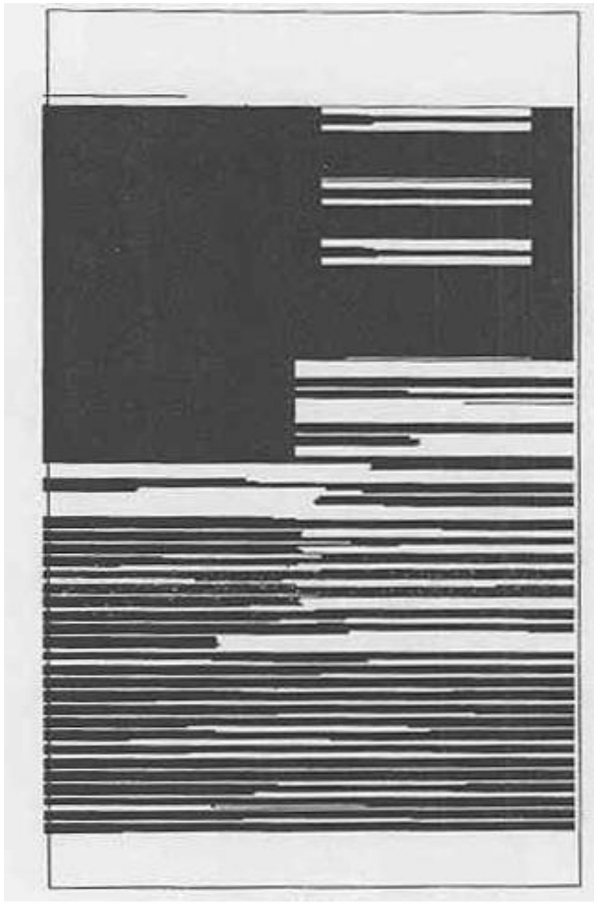
Text/Graphics Segmentation

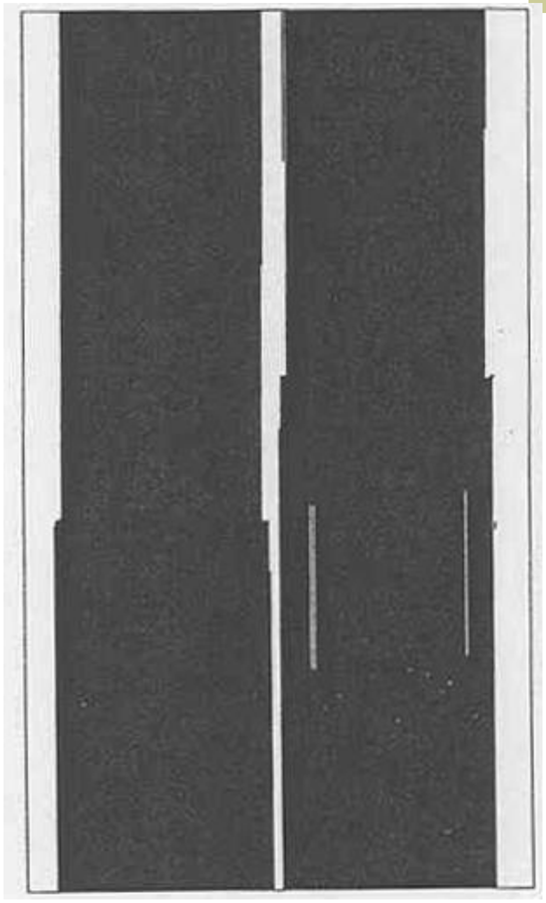




Vertical RLSA

Text/Graphics Segmentation

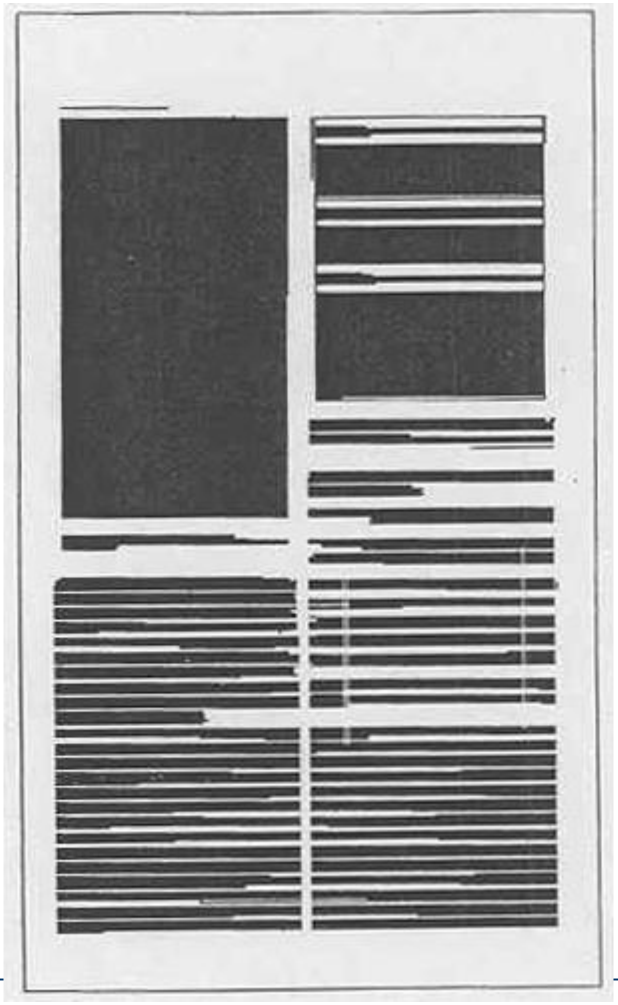

Horizontal RLSA


Vertical RLSA

Text/Graphics Segmentation

RLSA - Algorithm

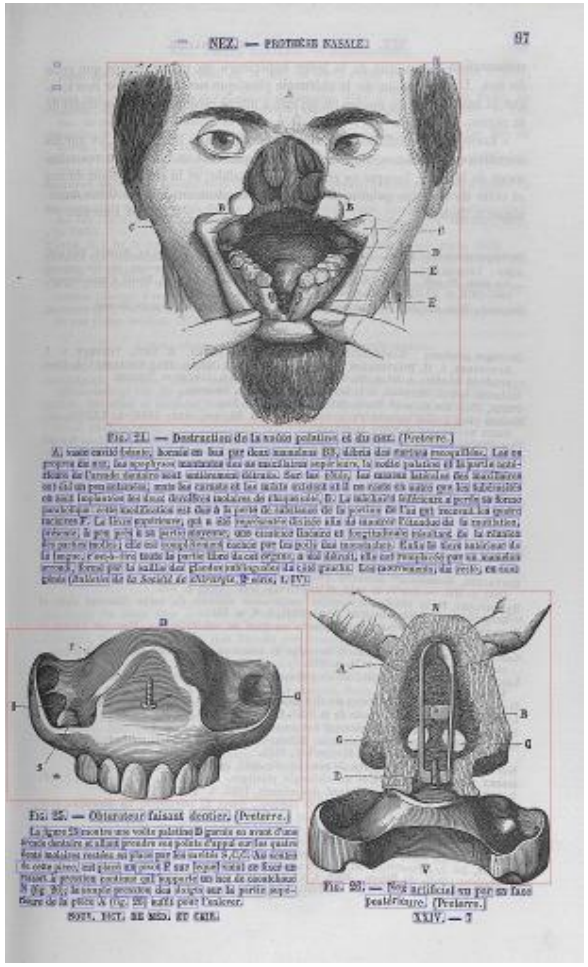
- Result of ANDING horizontal and vertical RLS-ed images



Text/Graphics Segmentation

RLSA - Algorithm

- Chose RLSA threshold to join characters instead of words



NEZ. — PROTHÈSE NASALE. 97

FIG. 24. — Destruction de la voûte palatine et du nez. (Preterre.)

[illegible]

FIG. 25. — Obturateur faisant dentier. (Preterre.)

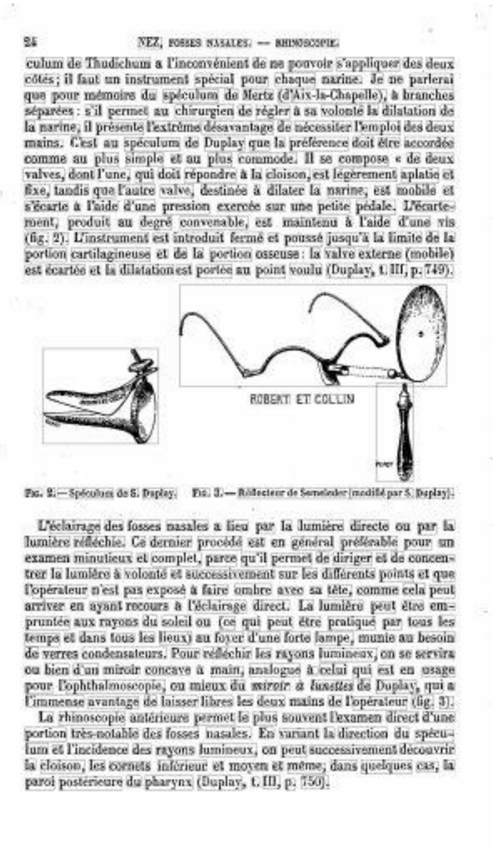
[illegible]

NOUV. DICT. DE MÉD. ET CHIR.

FIG. 26. — Nez artificiel vu par sa face postérieure. (Preterre.)

XXIV. — 7

Text/Graphics Segmentation



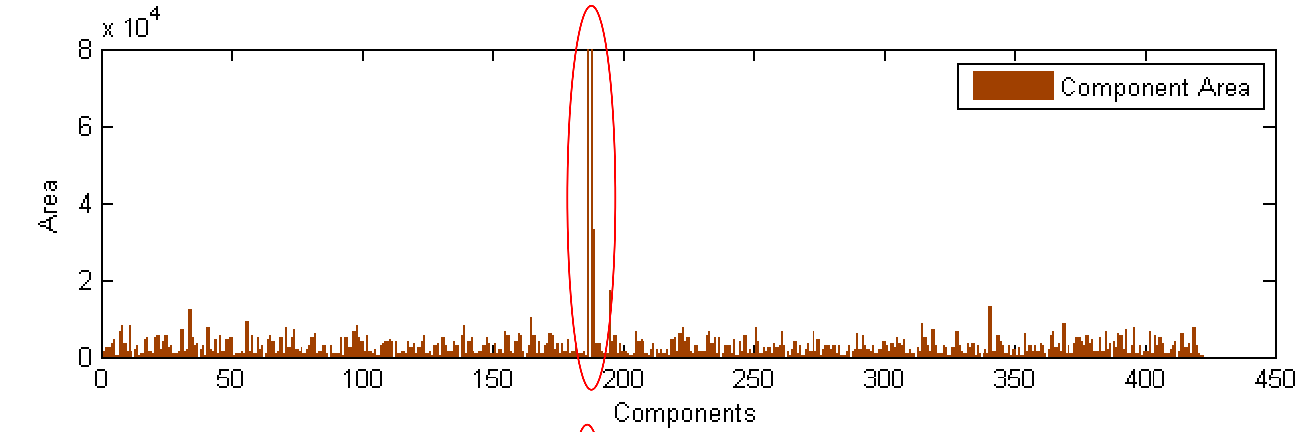
24 NEZ, FOSSES NASALES. — RHINOSCOPIE.

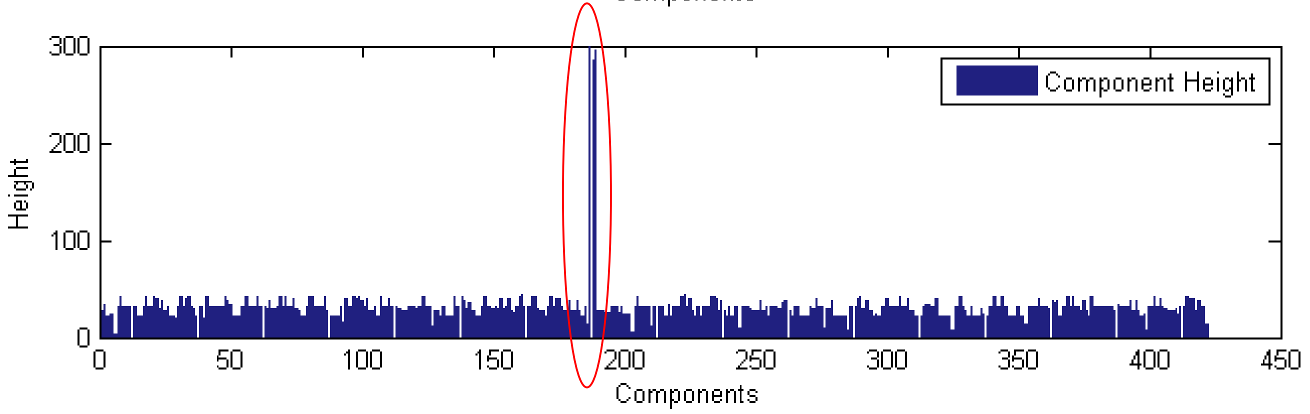
culum de Thudichum a l'inconvénient de ne pouvoir s'appliquer des deux côtés; il faut un instrument spécial pour chaque narine. Je ne parlerai que pour mémoire du spéculum de Mertz (d'Aix-la-Chapelle), à branches séparées : s'il permet au chirurgien de régler à sa volonté la dilatation de la narine, il présente l'extrême désavantage de nécessiter l'emploi des deux mains. C'est au spéculum de Duplay que la préférence doit être accordée comme au plus simple et au plus commode. Il se compose « de deux valves, dont l'une, qui doit répondre à la cloison, est légèrement aplatie et fixe, tandis que l'autre valve, destinée à dilater la narine, est mobile et s'écarte à l'aide d'une pression exercée sur une petite pédale. L'écartement, produit au degré convenable, est maintenu à l'aide d'une vis (fig. 2). L'instrument est introduit fermé et poussé jusqu'à la limite de la portion cartilagineuse et de la portion osseuse : la valve externe (mobile) est écartée et la dilatation est portée au point voulu (Duplay, t. III, p. 749).

FIG. 2. — Spéculum de S. Duplay. FIG. 3. — Réflecteur de Semeleder (modifié par S. Duplay).

L'éclairage des fosses nasales a lieu par la lumière directe ou par la lumière réfléchie. Ce dernier procédé est en général préférable pour un examen minutieux et complet, parce qu'il permet de diriger et de concentrer la lumière à volonté et successivement sur les différents points et que l'opérateur n'est pas exposé à faire ombre avec sa tête, comme cela peut arriver en ayant recours à l'éclairage direct. La lumière peut être empruntée aux rayons du soleil ou (ce qui peut être pratiqué par tous les temps et dans tous les lieux) au foyer d'une forte lampe, munie au besoin de verres condensateurs. Pour réfléchir les rayons lumineux, on se servira ou bien d'un miroir concave à main, analogue à celui qui est en usage pour l'ophthalmoscopie, ou mieux du *miroir à lunettes* de Duplay, qui a l'immense avantage de laisser libres les deux mains de l'opérateur (fig. 3).

La rhinoscopie antérieure permet le plus souvent l'examen direct d'une portion très-notable des fosses nasales. En variant la direction du spéculum et l'incidence des rayons lumineux, on peut successivement découvrir la cloison, les cornets inférieur et moyen et même, dans quelques cas, la paroi postérieure du pharynx (Duplay, t. III, p. 750).

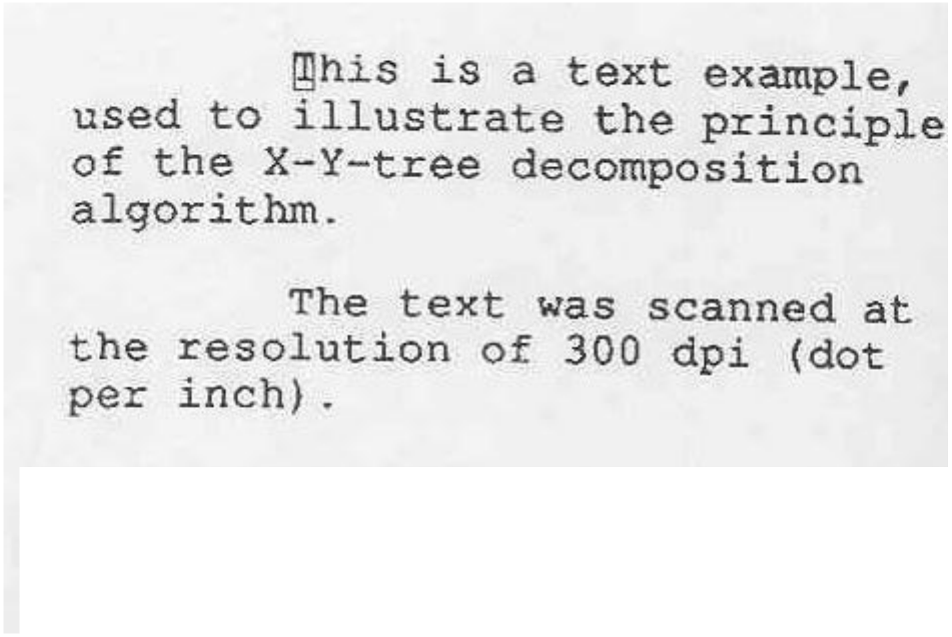




Line/Word Extraction

Line and Word Extraction

Line Extraction?

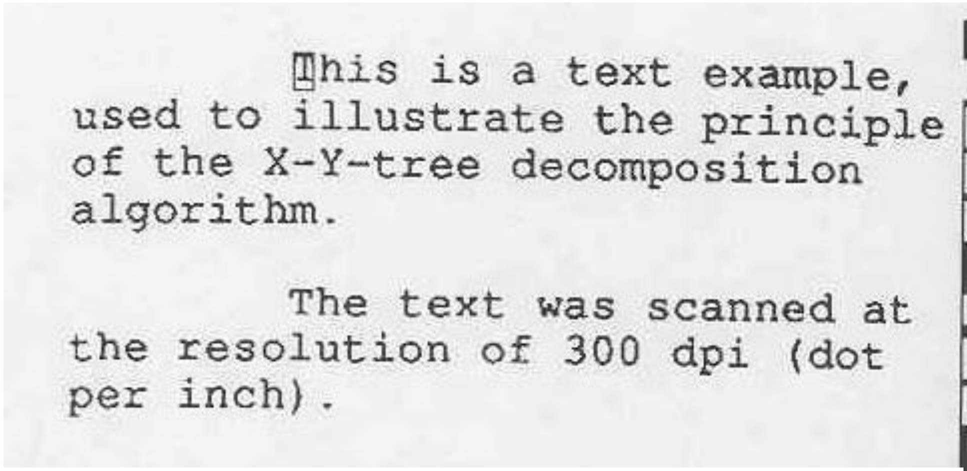


This is a text example, used to illustrate the principle of the X-Y-tree decomposition algorithm.

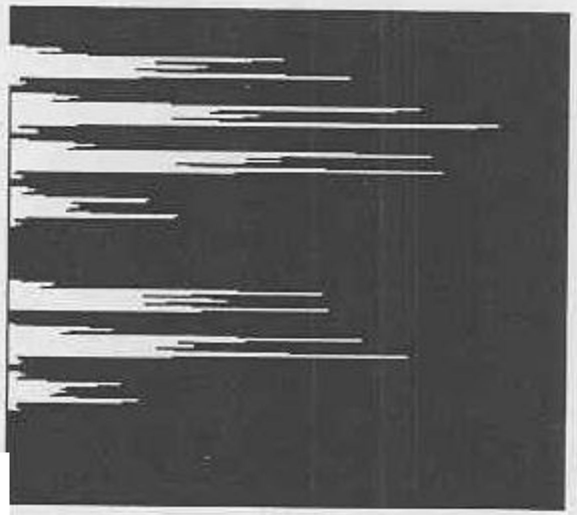
The text was scanned at the resolution of 300 dpi (dot per inch).

Line and Word Extraction

Line Extraction?

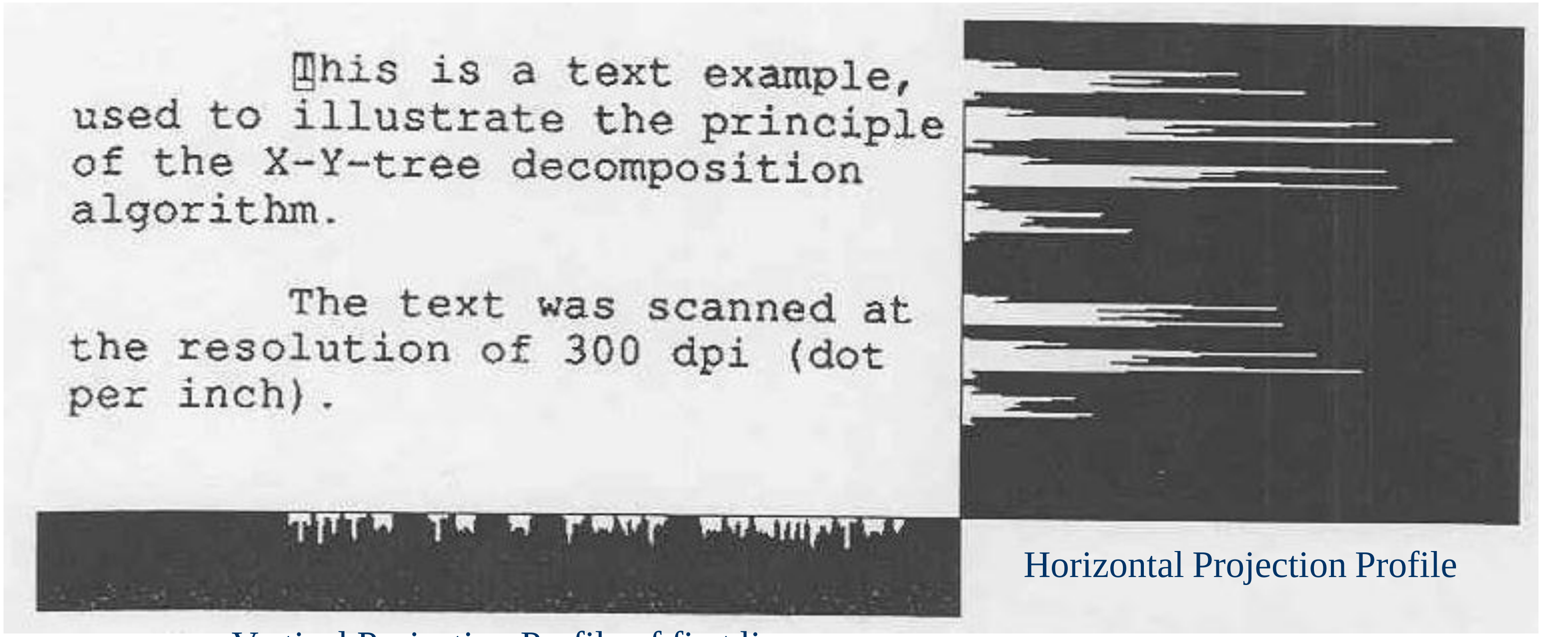

This is a text example, used to illustrate the principle of the X-Y-tree decomposition algorithm.

The text was scanned at the resolution of 300 dpi (dot per inch).


Horizontal Projection Profile

Line and Word Extraction

Projection Profiles



This is a text example, used to illustrate the principle of the X-Y-tree decomposition algorithm.

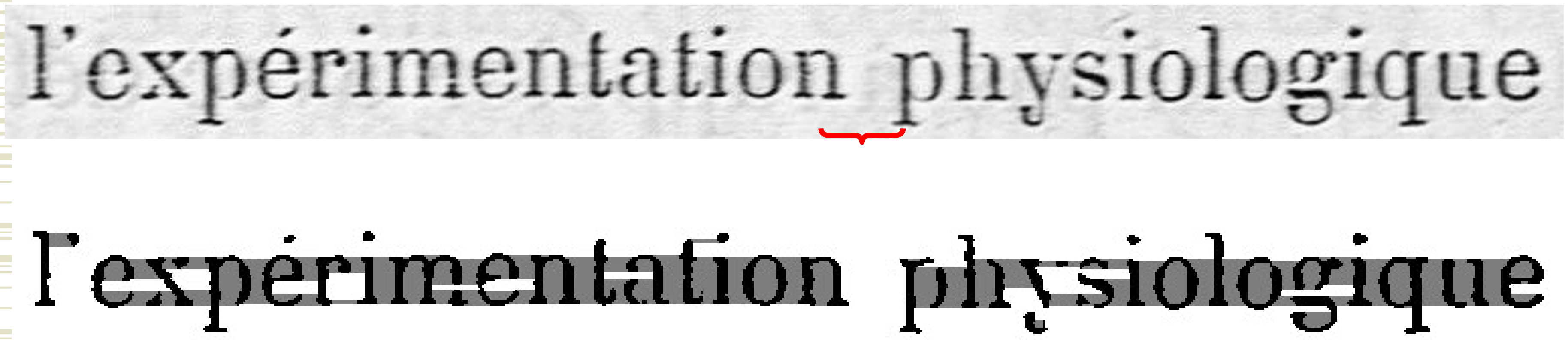
The text was scanned at the resolution of 300 dpi (dot per inch).

Horizontal Projection Profile

Vertical Projection Profile of first line

Word Extraction

Word Extraction using RLSA


l'expérimentation physiologique
l'expérimentation physiologique

Hit-or-Miss Transform

- A tool for shape detection or for the detection of a *disjoint region* in an image

- Idea
  - Suppose we have a binary image that contains certain shapes (circles, squares, lines, etc,….) called image A

  - We use another image or matrix to search image A for a particular pattern of bits. We will call this pattern “shape B”

  - We then search image A for shape B

  - Whenever there is a ‘hit’, we indicate where the center of shape B was on image A.

Hit-or-Miss Transform

- A tool for shape detection or for the detection of a *disjoint region* in an image
- Idea
  - Sup
    (cir
  - We
    particular pattern of bits. We will call this pattern "shape B"
  - We then search image A for shape B
  - Whenever there is a 'hit', we indicate where the center of shape B was on image A.

Watch out: We actually look for '*fits*' but we will be calling them '*hits*' when talking about hit-or-miss transform

Hit-or-Miss Transform

- Structuring Element

So far we have considered the SEs where 0s are treated as Don’t Cares i.e. we focus on the 1s only

$$\begin{array}{|c|c|c|}\hline & 1 & \\ \hline 1 & \boxed{1} & 1 \\ \hline & 1 & \\ \hline\end{array} \Rightarrow \begin{array}{|c|c|c|}\hline 0 & 1 & 0 \\ \hline 1 & \boxed{1} & 1 \\ \hline 0 & 1 & 0 \\ \hline\end{array}$$

Hit-or-Miss Transform

- Extended Structuring Element

Now we will distinguish between the 0s and the Don't cares

1	1	1
×	0	×
×	0	×

E.g. For a 'fit' the 0s of SE should match with 0s of the underlying image

Hit-or-Miss Transform

- Extended Structuring Element: Example

0	0	0	0	0	0	0	0
0	0	1	1	1	0	0	0
0	0	0	0	1	1	1	0
0	1	0	0	1	1	1	1
0	1	1	1	1	1	0	1
0	0	1	1	1	1	0	1

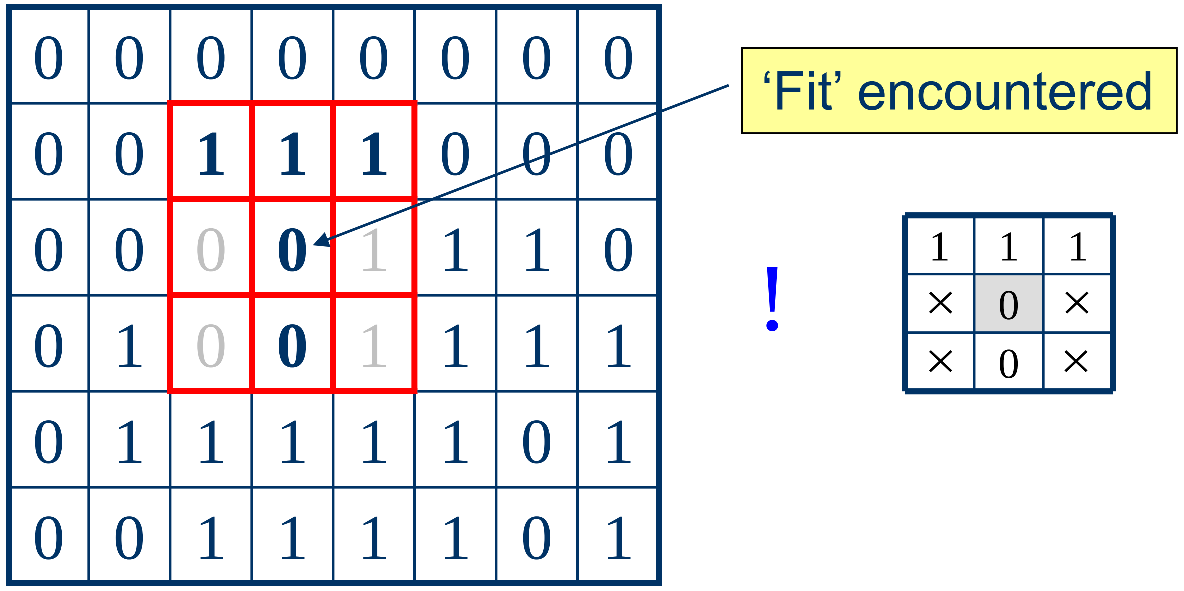
!

1	1	1
×	0	×
×	0	×

Erosion Recap: Slide the SE on the image and look for the ‘fits’

Hit-or-Miss Transform

- Extended Structuring Element: Example



0	0	0	0	0	0	0	0
0	0	**1**	**1**	**1**	0	0	0
0	0	0	**0**	1	1	1	0
0	1	0	**0**	1	1	1	1
0	1	1	1	1	1	0	1
0	0	1	1	1	1	0	1

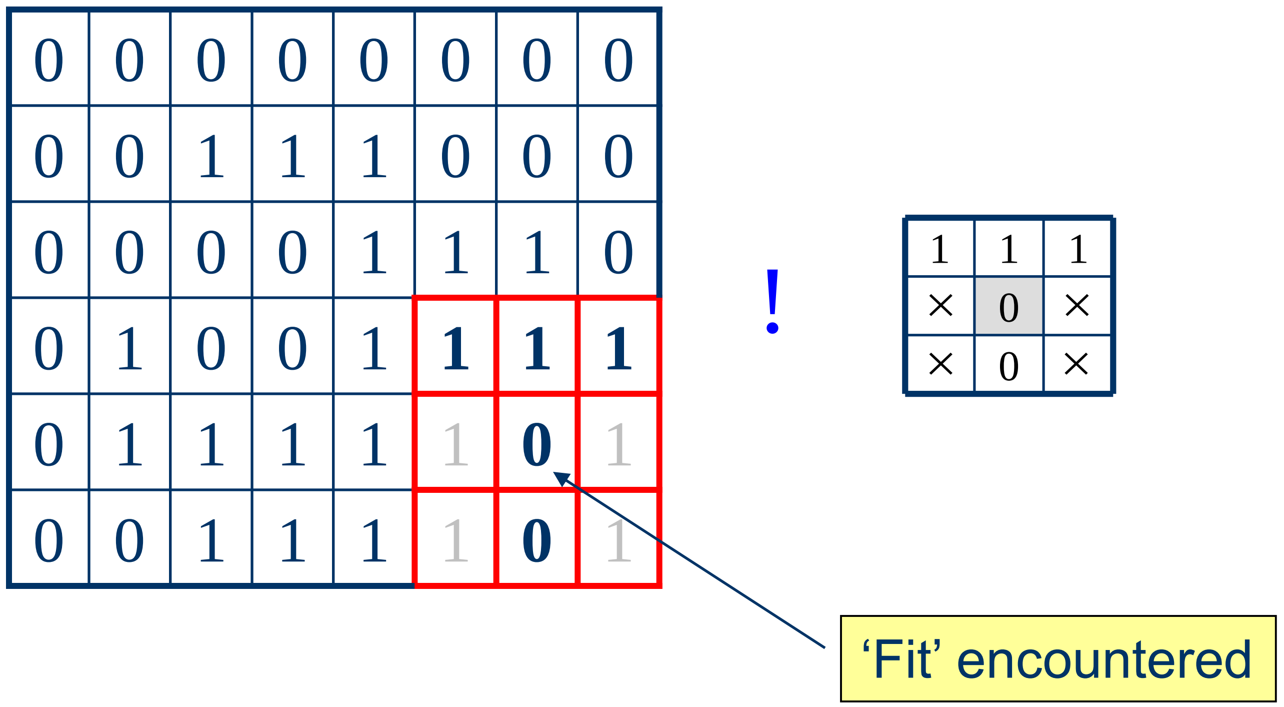
'Fit' encountered

!

1	1	1
×	0	×
×	0	×

Hit-or-Miss Transform

- Extended Structuring Element: Example



0	0	0	0	0	0	0	0
0	0	1	1	1	0	0	0
0	0	0	0	1	1	1	0
0	1	0	0	1	**1**	**1**	**1**
0	1	1	1	1	1	**0**	1
0	0	1	1	1	1	**0**	1

!

1	1	1
×	0	×
×	0	×

Hit-or-Miss Transform

- Extended Structuring Element: Example

$$\begin{array}{|c|c|c|c|c|c|c|c|}\hline 0&0&0&0&0&0&0&0\\\hline 0&0&1&1&1&0&0&0\\\hline 0&0&0&0&1&1&1&0\\\hline 0&1&0&0&1&1&1&1\\\hline 0&1&1&1&1&1&0&1\\\hline 0&0&1&1&1&1&0&1\\\hline\end{array} \; ! \; \begin{array}{|c|c|c|}\hline 1&1&1\\\hline \times&0&\times\\\hline \times&0&\times\\\hline\end{array} \; = \; \begin{array}{|c|c|c|c|c|c|c|c|}\hline 0&0&0&0&0&0&0&0\\\hline 0&0&0&0&0&0&0&0\\\hline 0&0&0&\mathbf{1}&0&0&0&0\\\hline 0&0&0&0&0&0&0&0\\\hline 0&0&0&0&0&0&\mathbf{1}&0\\\hline 0&0&0&0&0&0&0&0\\\hline\end{array}$$

What we actually did???

Hit-or-Miss Transform

We have searched the pattern in the structuring element in the image

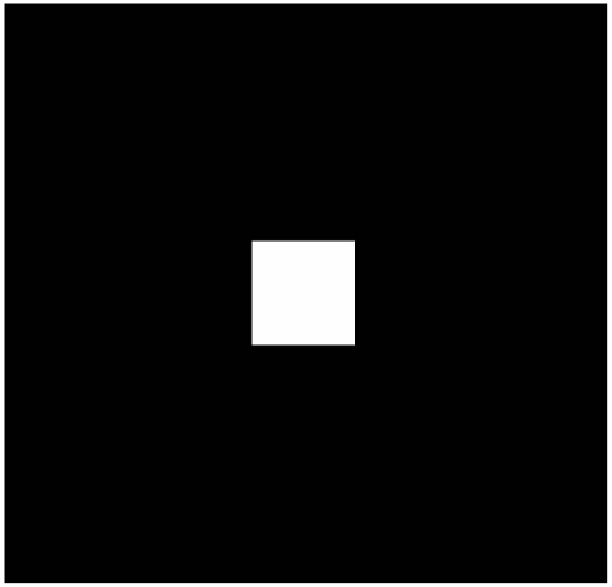
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	**1**	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	**1**	0
0	0	0	0	0	0	0	0

Output: The center of the pattern is 1 and rest everything is 0

Hit-or-Miss Transform

Example: Search a 100x100 pixel square in an image

- How do we search?
  - Take an image of size 100x100 (B) representing a white square
  - We search the pattern in the input image (A)
  - If found, we have a “fit”. We mark the center of the “fit” with a white pixel
  - In the above example, there would be only 1 fit

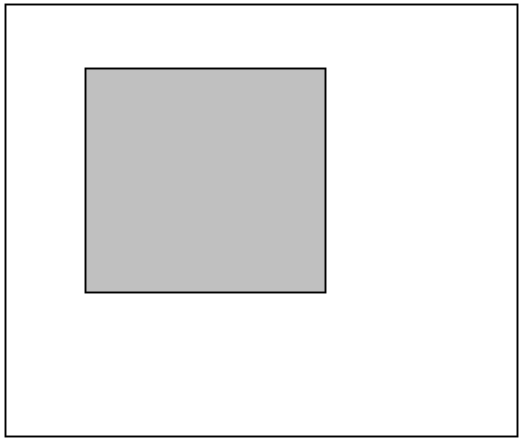


$$\boxed{(A!\ \ B)}$$

Hit-or-Miss Transform

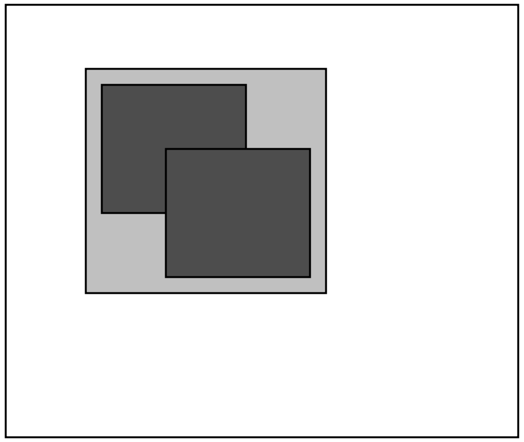
Do you find any problem with this?

If we search for a 100x100 pixel square in an image we will have a positive response for all squares greater than 100x100 as well



Image

Shape to pattern


The pattern will fit at different places

$(A!\ \ B)\cap$ Need some thing here

Hit-or-Miss Transform

We will limit our discussion to this simple version only

$$A\#\ B = (A!\ \ B)$$

Morphological Algorithms

Using the simple technique we have looked at so far we can begin to consider some more interesting morphological algorithms

We will look at:

- Boundary extraction
- Region filling
- Extraction of connected components

There are lots of others as well though:

- Thinning/thickening
- Skeletonization

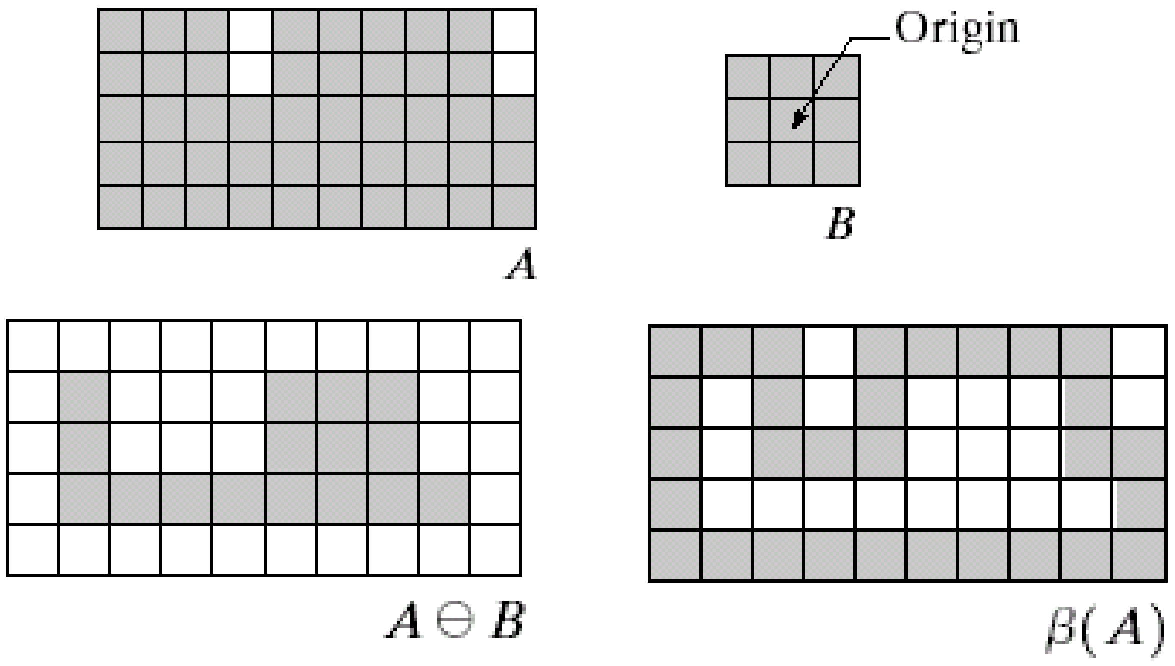
Boundary Extraction

The boundary of set A denoted by *β(A)* is obtained by first eroding *A* by a suitable structuring element *B* and then taking the difference between *A* and its erosion.

$$\beta(A) = A - (A \ominus B)$$

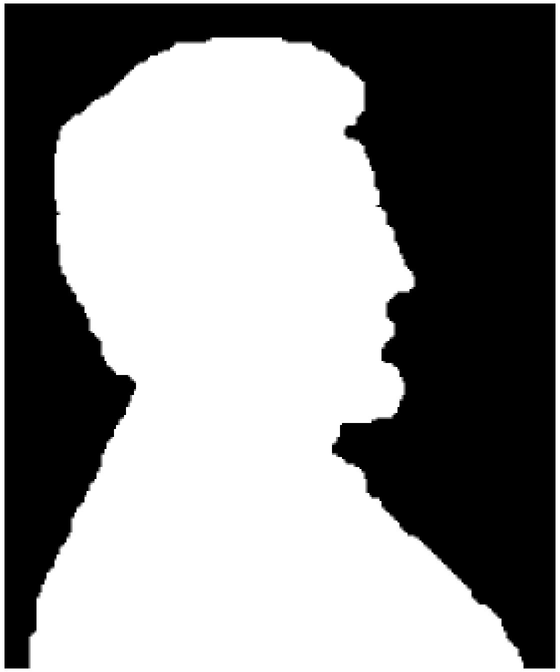
Boundary Extraction

$$\beta(A) = A - (A\,!\;\;B)$$

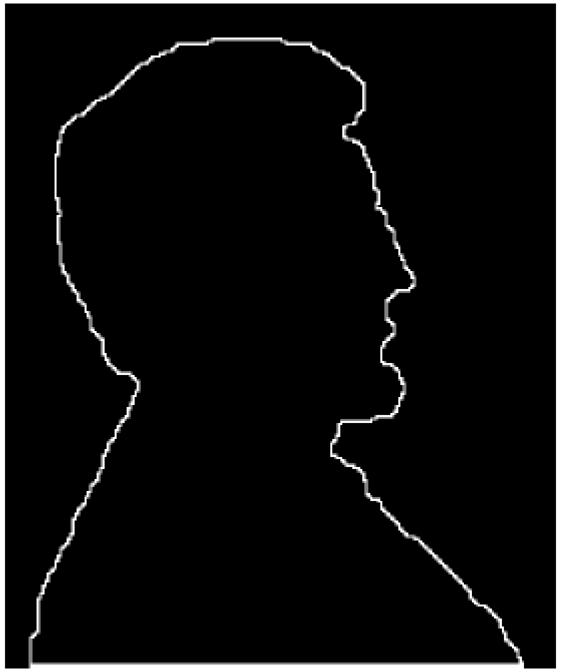


Boundary Extraction

A simple image and the result of performing boundary extraction using a square 3*3 structuring element



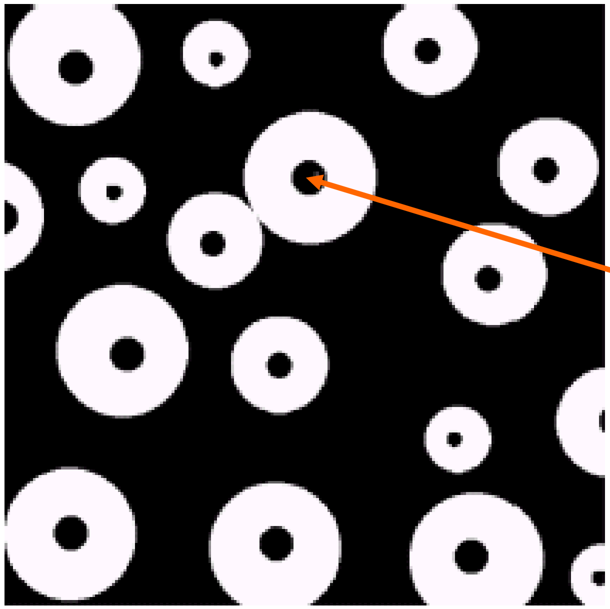
Original Image



Extracted Boundary

Region (hole) Filling

Given a pixel inside a boundary, *region filling* attempts to fill that boundary with object pixels (1s)



Given a point inside here, can we fill the whole circle?

Region Filling

Let A is a set containing a subset whose elements are 8-connected boundary points of a region, enclosing a background region i.e. hole

If all boundary points are labeled 1 and non boundary points are labeled 0, the following procedure fills the region:

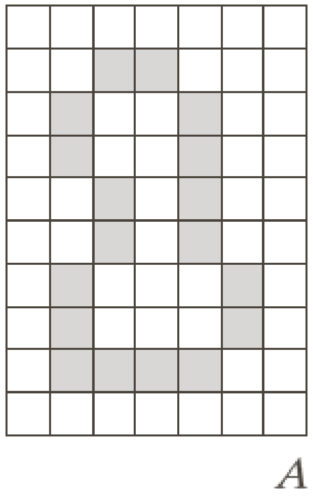
Inside the boundary

- Start from a known point p and taking $X_0 = p$,
- Then taking the next values of X_k as:

$$X_k = (X_{k-} \oplus B) \bigcap A^c \qquad k = 1, 2, 3, \cdots$$

B is suitable structuring element

- Terminate iterations if $X_{k+1} = X_k$
- The set union of X_k and A contains the filled set and its boundaries.



Region Filling

0	0	0	0	0	0	0
0	1	1	1	1	1	0
0	1	0	0	0	1	0
0	1	0	0	0	1	0
0	1	0	0	0	1	0
0	1	1	1	1	1	0
0	0	0	0	0	0	0

A

1	1	1	1	1	1	1
1	0	0	0	0	0	1
1	0	1	1	1	0	1
1	0	1	1	1	0	1
1	0	1	1	1	0	1
1	0	0	0	0	0	1
1	1	1	1	1	1	1

A^c

0	1	0
1	1	1
0	1	0

B

Region Filling

0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	1	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0

X_0

0	1	0
1	1	1
0	1	0

B

0	0	0	0	0	0	0
0	0	1	0	0	0	0
0	1	1	1	0	0	0
0	0	1	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0

$(X_0 \oplus B)$

$$X_k = (X_{k-} \oplus B) \cap A^c$$

$$k = 1, 2, 3, \cdots$$

Region Filling

0	0	0	0	0	0	0
0	0	1	0	0	0	0
0	1	1	1	0	0	0
0	0	1	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0

$X_1 = (X_0 \oplus B)$

1	1	1	1	1	1	1
1	0	0	0	0	0	1
1	0	1	1	1	0	1
1	0	1	1	1	0	1
1	0	1	1	1	0	1
1	0	0	0	0	0	1
1	1	1	1	1	1	1

A^c

0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	1	1	0	0	0
0	0	1	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0

$X_1 = (X_0 \oplus B) \bigcap A^c$

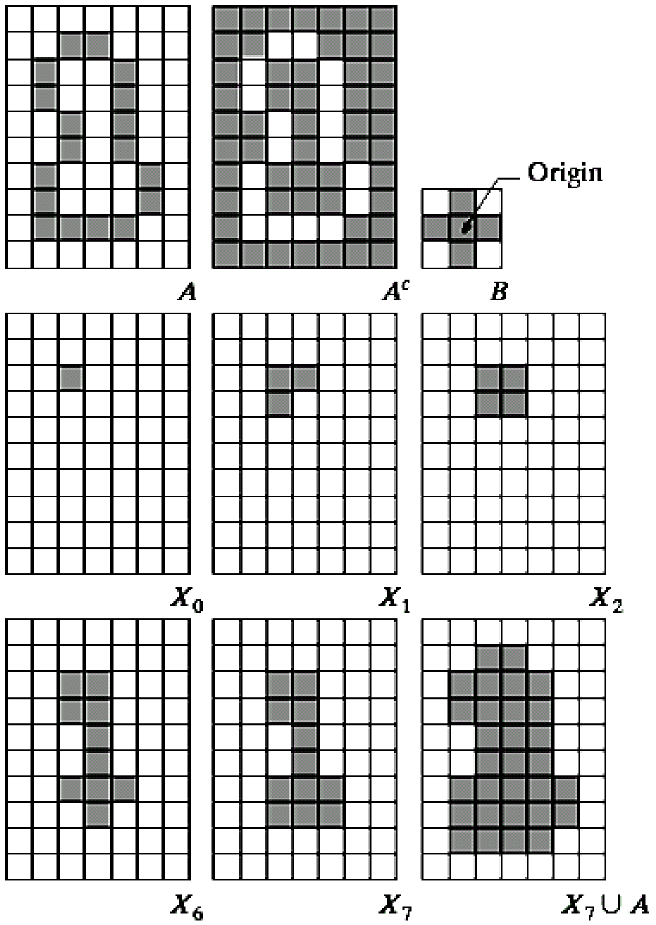
$$X_k = (X_{k-} \oplus B) \bigcap A^c$$

$$k = 1, 2, 3, \cdots$$

Region Filling

$$X_k = (X_{k-1} \oplus B) \cap A^c$$
$$k = 1, 2, 3, \cdots$$

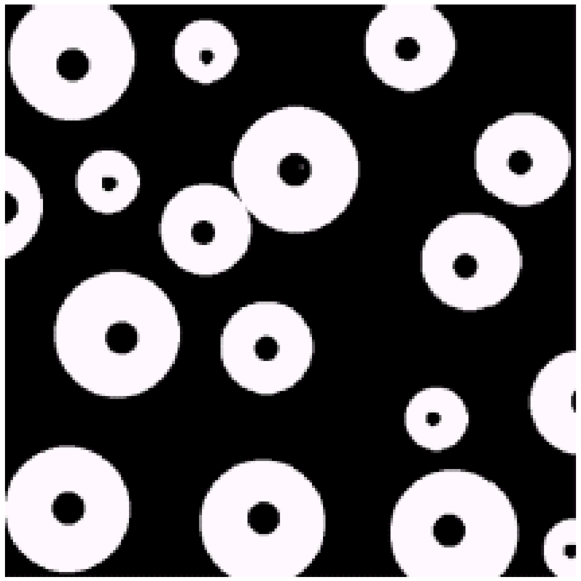
NOTE:
The intersection of dilation and the complement of A limits the result to inside the region of interest

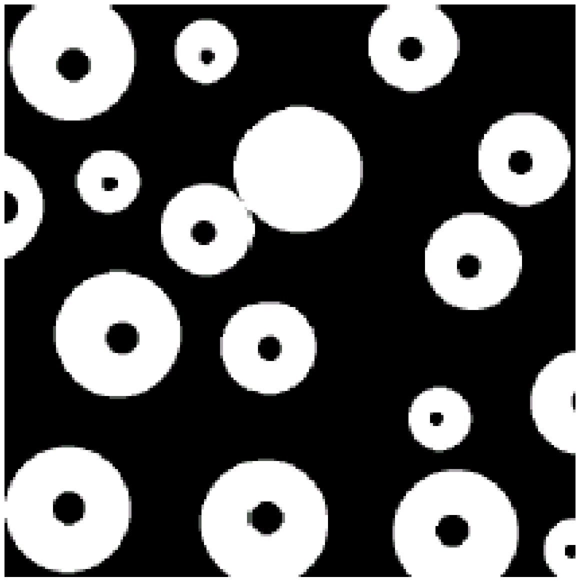


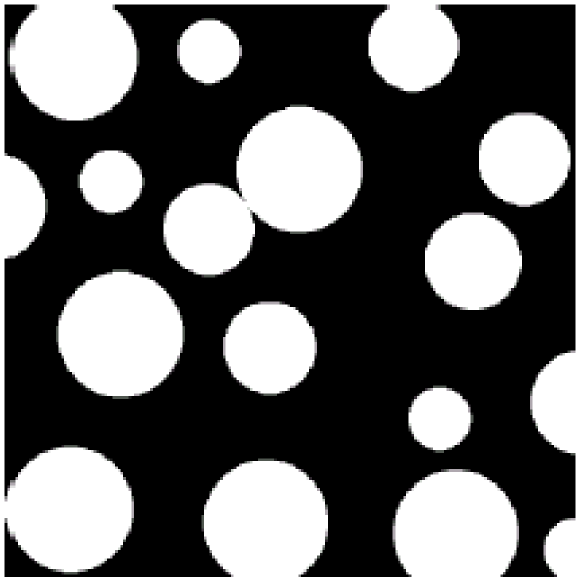
a	b	c
d	e	f
g	h	i

FIGURE 9.15 Hole filling. (a) Set A (shown shaded). (b) Complement of A. (c) Structuring element B. (d) Initial point inside the boundary. (e)–(h) Various steps of Eq. (9.5-2). (i) Final result [union of (a) and (h)].

Region Filling: Example


Original Image


One Region Filled


All Regions Filled

Extraction of Connected Components (CCs)

Let Y represents a connected component contained in *A* and the point *p* of the Y is known.

The following procedure iteratively finds all the elements of Y:

- Start from a known point *p* and taking $X_0 = p$,
- Then taking the next values of X_k as:

$$X_k = (X_{k-1} \oplus B) \bigcap A \qquad k = 1, 2, 3, \cdots$$

B is a suitable structuring element

- Algorithm converges if $X_k = X_{k-1}$
- The component Y is given as $Y = X_k$

Extraction of CCs: Example

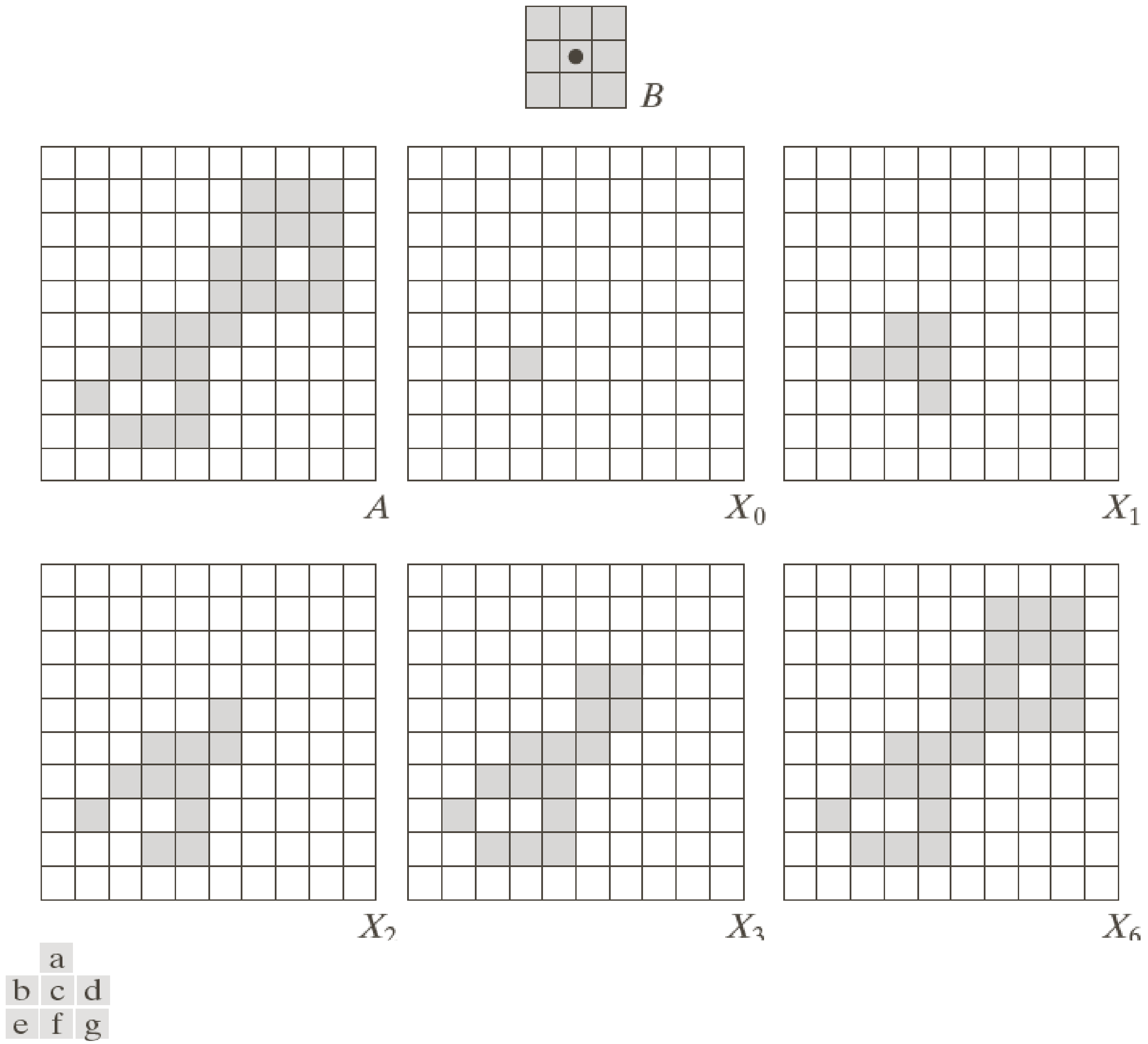
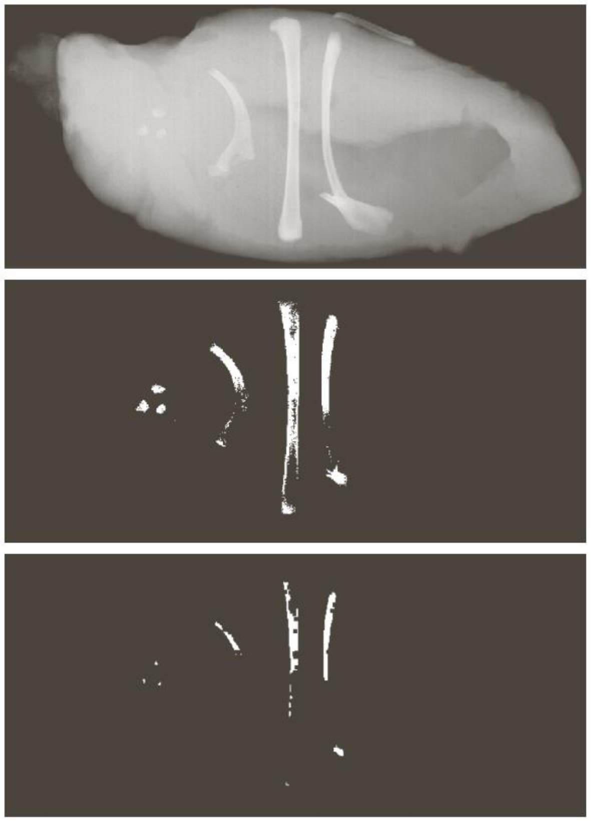


FIGURE 9.17 Extracting connected components. (a) Structuring element. (b) Array containing a set with one connected component. (c) Initial array containing a 1 in the region of the connected component. (d)–(g) Various steps in the iteration of Eq. (9.5-3).

Extraction of CCs: Example



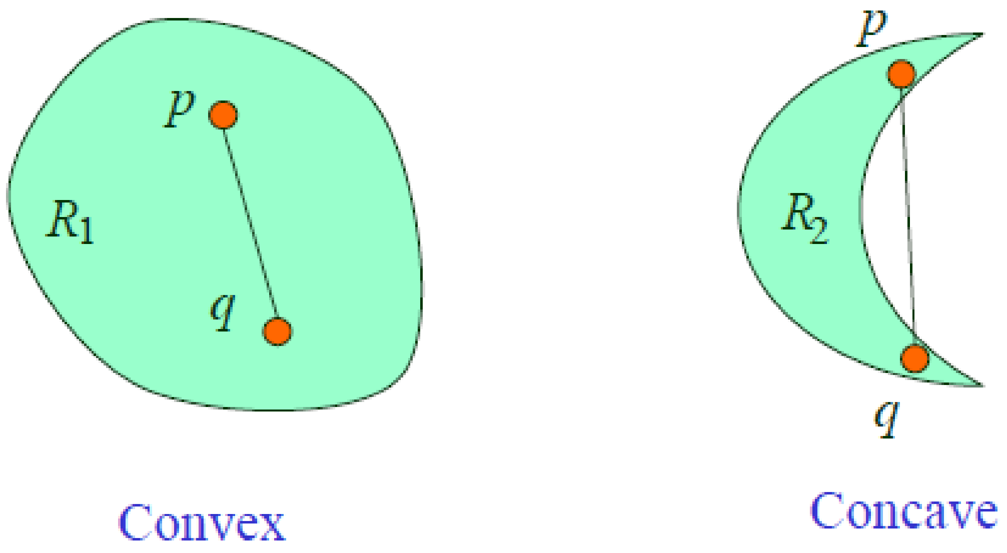
Connected component	No. of pixels in connected comp
01	11
02	9
03	9
04	39
05	133
06	1
07	1
08	743
09	7
10	11
11	11
12	9
13	9
14	674
15	85

a
b
c d

FIGURE 9.18 (a) X-ray image of chicken filet with bone fragments. (b) Thresholded image. (c) Image eroded with a 5 × 5 structuring element of 1s. (d) Number of pixels in the connected components of (c). (Image courtesy of NTB Elektronische Geraete GmbH, Diepholz, Germany, www.ntbxray.com.)

Convex Hull

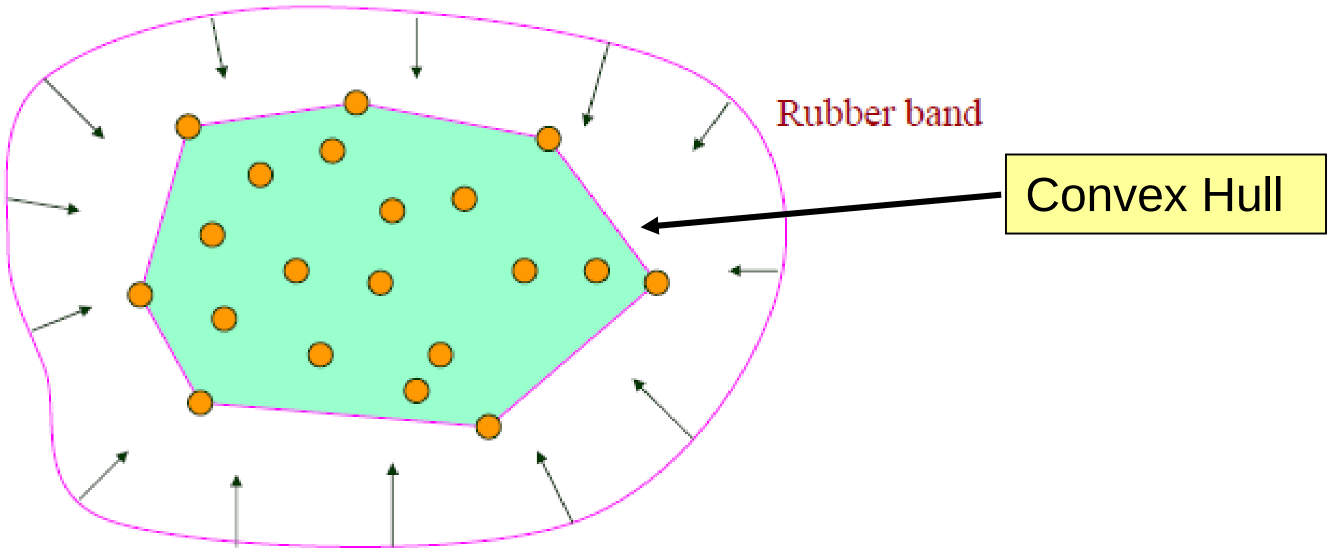
- Convex set
  A set A is said to be convex if any straight line segment joining two points of A lies within A
- Example: R_1 is convex as line segment pq lies within set R_1



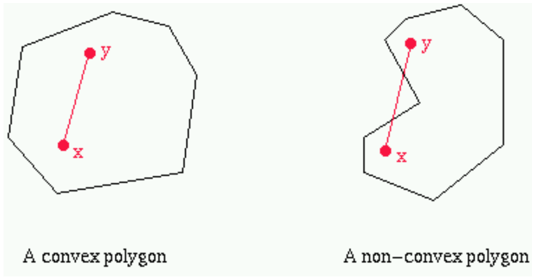
Convex Hull

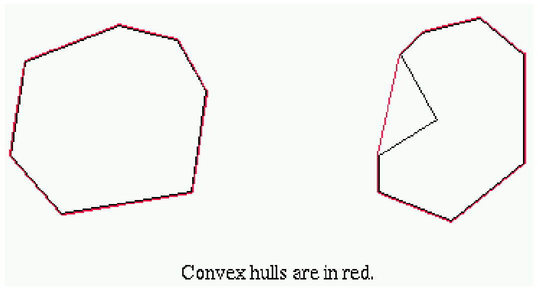
- Convex Hull

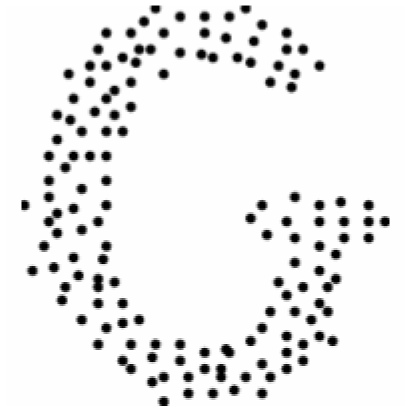
  Convex hull H of a set S is the smallest convex set containing S
- Rubber band example:

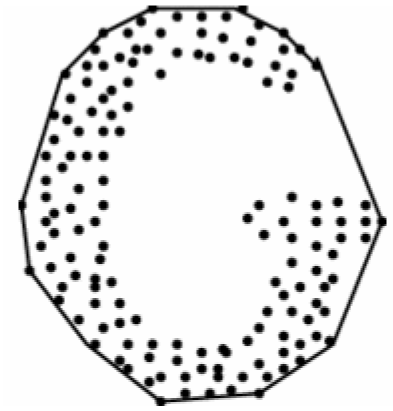


Convex Hull


y
x
A convex polygon
y
x
A non–convex polygon


Convex hulls are in red.





Convex Hull

- To find the Convex Hull *C(A)* of a set *A* the following simple morphological algorithm can be used:

- Let B^i, where $i = 1, 2, 3, 4$, represent four structuring elements
- Implement:

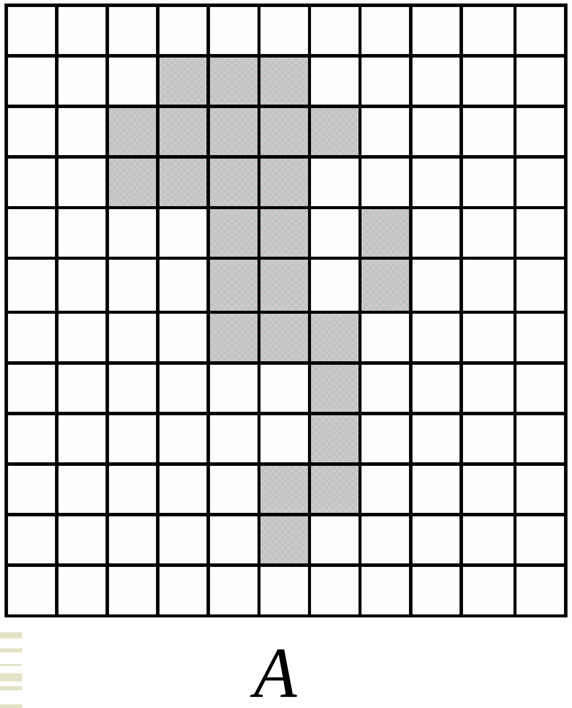
$$\boxed{X^i_k = (X^i_{k-1} \circledast B^i) \cup A \quad i = 1,2,3,4 \quad and \quad k = 1,2,3,\cdots}$$

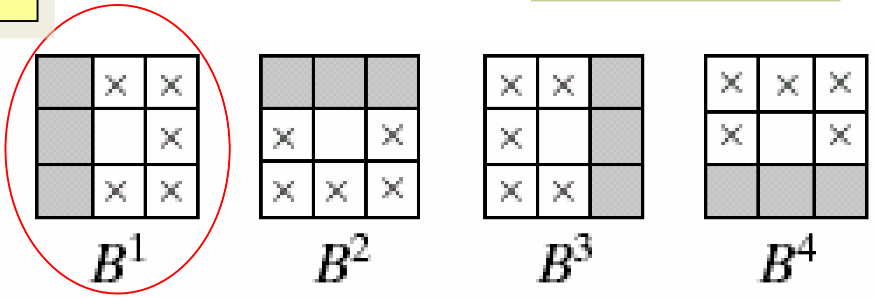
- Starting with: $X^i_0 = A$
- Repeat 2nd step until convergence, i.e. $D^i = X^i_{conv} \rightarrow X^i_k = X^i_{k+1}$
- Convex Hull *C(A)* is given by:

$$\boxed{C(A) = \bigcup_{i=1}^{4} D^i}$$

Convex Hull

Pick the first SE





Start At:

$$X_0^1 = A$$

Find:

$$X_1^1 = (X_0^1 \# B^1) \cup A$$

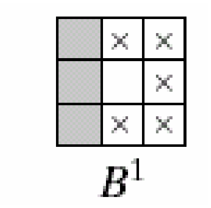
$$X_2^1 = (X_1^1 \# B^1) \cup A$$

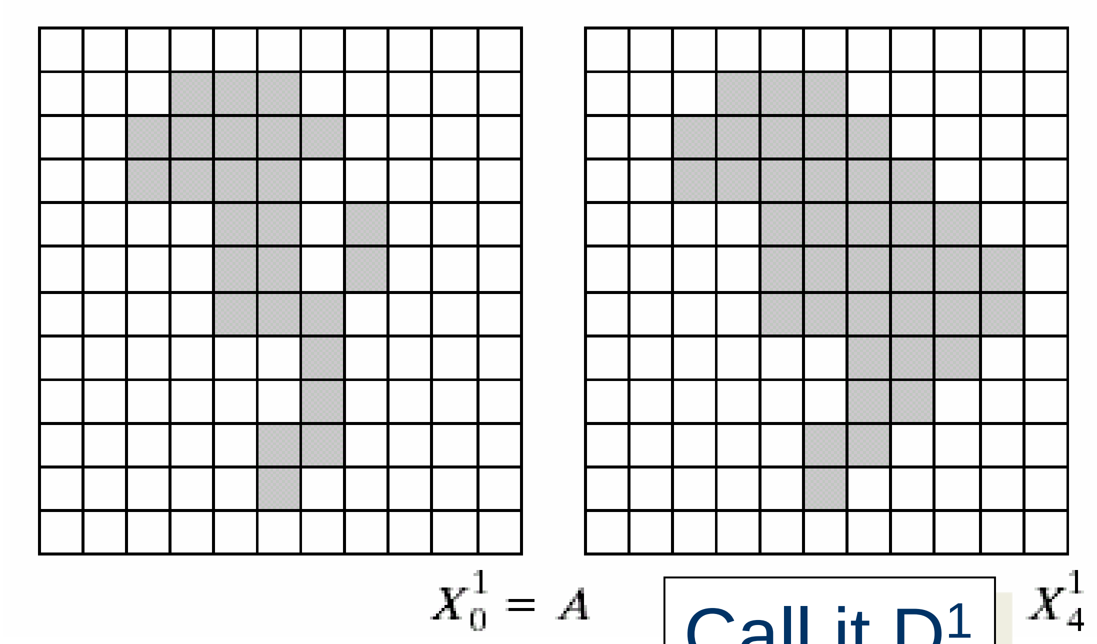
$$\vdots$$

Until Convergence $X_k^1 = (X_{k-1} \# B^1) \cup A$

Convex Hull

$$X^1_k = (X_{k-1} \# B^1) \cup A$$



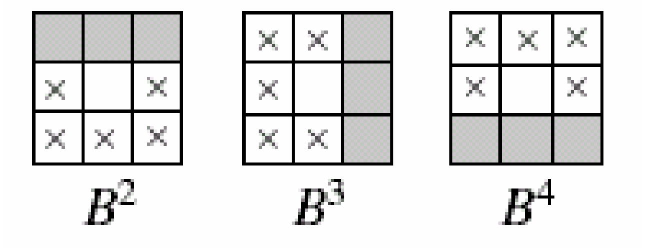


Call it D^1

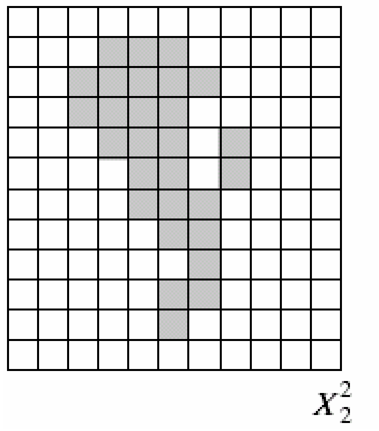
Convergence after four iterations

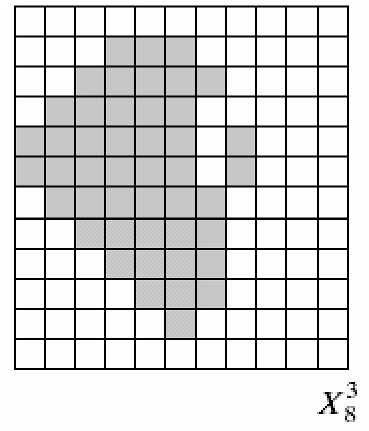
Convex Hull

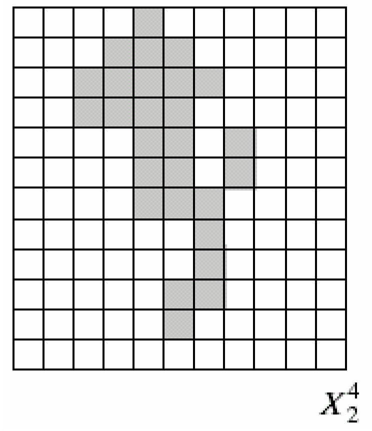
Repeat the same process for B², B³ and B⁴



$$X^i_k = (X_{k-1} \# B^i) \cup A \quad i = 2,3,4$$





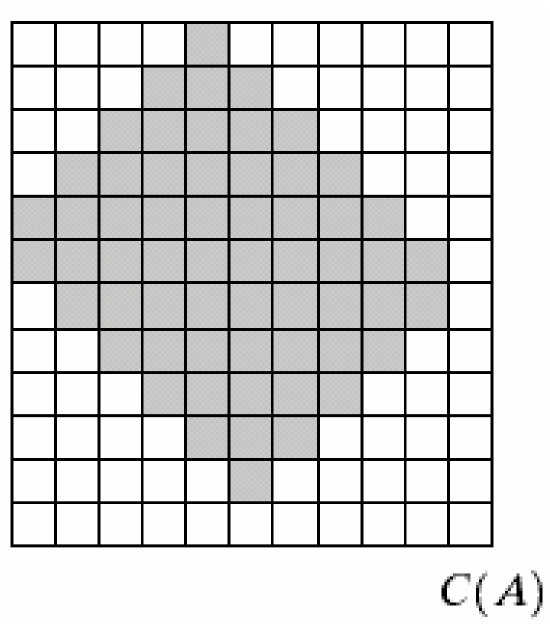


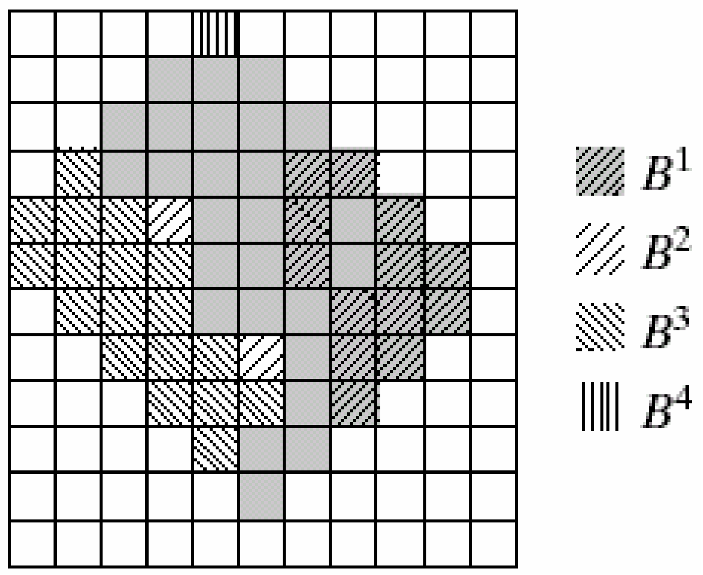
D² D³ D⁴

Convex Hull

Take the union of all D^i to get the convex hull of A

$$C(A) = \bigcup_{i=1}^{4} D^i$$





Acknowledgements

- Digital Image Processing", Rafael C. Gonzalez & Richard E. Woods, Addison-Wesley, 2002
- Peters, Richard Alan, II, Lectures on Image Processing, Vanderbilt University, Nashville, TN, April 2008
- Brian Mac Namee, Digitial Image Processing, School of Computing, Dublin Institute of Technology
- Computer Vision for Computer Graphics, Mark Borg

Material in these slides has been taken from, the following resources