Embedded Vision Design

EVD1 - Week 5

Morphological Filters

By Hugo Arends

HAN_UNIVERSITY OF APPLIED SCIENCES

Morphological filters

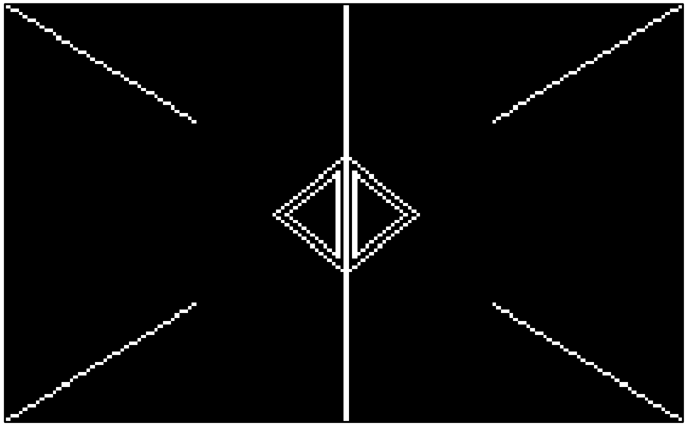
- Are used prior to pattern recognition and object classification
- Changes the geometrical shape of the objects
- The goal is to smooth the object's contours and to decompose objects in their fundamental shapes

- Remove border blobs
- Fill holes
- Dilation & Erosion
- Closing & Opening
- Hit-miss
- Outline
- Skeleton

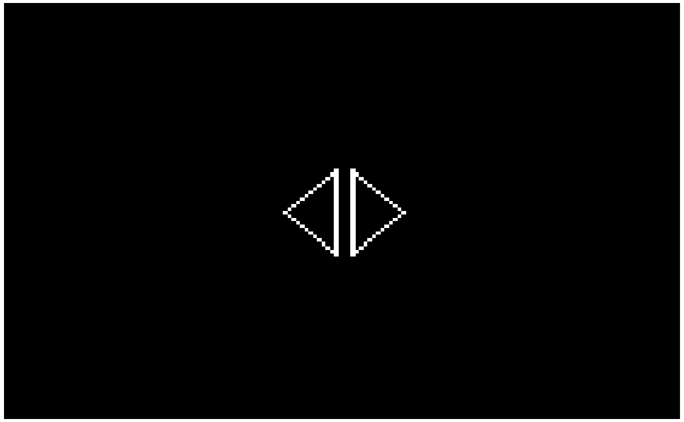
Remove border BLOBs

- Removes all objects that are 4/8-connected to a border

Remove border BLOBs - example



4-connected



8-connected

Remove border BLOBs

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

Source

						1	
					1	1	
						1	
						1	

Destination 8-connected

Remove border BLOBs – Iterative algorithm

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

Remove border BLOBs – Iterative algorithm

		2					
	1					1	
					1	1	
	1		1			1	
	1	1	1			1	
	1						
	1				1	1	
	2		2	2	2	2	

- Mark border object pixels

Remove border BLOBs – Iterative algorithm

		2					
	2					1	
					1	1	
	1		1			1	
	1	1	1			1	
	1						
	1				1	1	
	2		2	2	2	2	

- Mark border object pixels
- While changes
  - Loop entire image and assign marker value if a neighbor is also marked

Remove border BLOBs – Iterative algorithm

		2					
	2					1	
					1	1	
	1		1			1	
	1	1	1			1	
	1						
	2				1	1	
	2		2	2	2	2	

- Mark border object pixels
- While changes
  - Loop entire image and assign marker value if a neighbor is also marked

Remove border BLOBs – Iterative algorithm

		2					
	2					1	
					1	1	
	1		1			1	
	1	1	1			1	
	1						
	2				2	1	
	2		2	2	2	2	

- Mark border object pixels
- While changes
  - Loop entire image and assign marker value if a neighbor is also marked

Remove border BLOBs – Iterative algorithm

		2					
	2					1	
					1	1	
	1		1			1	
	1	1	1			1	
	1						
	2			2	2		
	2		2	2	2	2	

- Mark border object pixels
- While changes
  - Loop entire image and assign marker value if a neighbor is also marked

Remove border BLOBs – Iterative algorithm

		2					
	2					1	
					1	1	
	1		1			1	
	1	1	1			1	
	2						
	2				2	2	
	2		2	2	2	2	

- Mark border object pixels
- While changes
  - Loop entire image and assign marker value if a neighbor is also marked

Remove border BLOBs – Iterative algorithm

		2					
	2					1	
					1	1	
	1		1			1	
	2	1	1			1	
	2						
	2				2	2	
	2		2	2	2	2	

- Mark border object pixels
- While changes
  - Loop entire image and assign marker value if a neighbor is also marked

Remove border BLOBs – Iterative algorithm

		2					
	2					1	
					1	1	
	1		1			1	
	2	2	1			1	
	2						
	2			2	2		
	2		2	2	2	2	

- Mark border object pixels
- While changes
  - Loop entire image and assign marker value if a neighbor is also marked

Remove border BLOBs – Iterative algorithm

		2					
	2					1	
					1	1	
	1		1			1	
	2	2	2			1	
	2						
	2				2	2	
	2		2	2	2	2	

- Mark border object pixels
- While changes
  - Loop entire image and assign marker value if a neighbor is also marked

Remove border BLOBs – Iterative algorithm

		2					
	2					1	
					1	1	
	2		1			1	
	2	2	2			1	
	2						
	2				2	2	
	2		2	2	2	2	

- Mark border object pixels
- While changes
  - Loop entire image and assign marker value if a neighbor is also marked

Remove border BLOBs – Iterative algorithm

		2					
	2					1	
					1	1	
	2		2			1	
	2	2	2			1	
	2						
	2				2	2	
	2		2	2	2	2	

- Mark border object pixels
- While changes
  - Loop entire image and assign marker value if a neighbor is also marked

Remove border BLOBs – Iterative algorithm

						1	
					1	**1**	
						1	
						1	

- Mark border object pixels
- While changes
  - Loop entire image and assign marker value if a neighbor is also marked
- Set marked pixels to background value (0)

Remove border BLOBs – Iterative algorithm

Advantage

- Easy implementation

Disadvantage

- Very slow, especially if the image is scanned in a single direction and the object happens the point towards the opposite direction

Remove border BLOBs – Iterative algorithm

uint32_t **removeBorderBlobsIterative**(
const image_t ***src**, image_t ***dst,**
const eConnected **connected**);

See file **EVDK_Operators\morphological_filters.c**

```
// Threshold the image
threshold2Means(src, tmp, BRIGHTNESS_DARK);

// Remove the border BLOBs
removeBorderBlobsIterative(tmp, dst, CONNECTED_EIGHT);
```

Remove border BLOBs – Two-pass algorithm

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

Equivalence LUT					
1					
1					

Remove border BLOBs – Two-pass algorithm

		2					
	1					1	
					1	1	
	1		1			1	
	1	1	1			1	
	1						
	1				1	1	
	2		2	2	2	2	

- Mark border object pixels
- Update equivalence LUT

Equivalence LUT					
1	2				
1	2				

Remove border BLOBs – Two-pass algorithm

		2					
	2					1	
					1	1	
	1		1			1	
	1	1	1			1	
	1						
	1				1	1	
	2		2	2	2	2	

- Mark border object pixels
- Update equivalence LUT
- Loop entire image
  - Is it an object pixel?
  - Is a neighbor already marked?
    - Yes: **mark this pixel with the same value**

Equivalence LUT					
1	2				
1	2				

Remove border BLOBs – Two-pass algorithm

		2					
	2					3	
					1	1	
	1		1			1	
	1	1	1			1	
	1						
	1				1	1	
	2		2	2	2	2	

- Mark border object pixels
- Update equivalence LUT
- Loop entire image
  - Is it an object pixel?
  - Is a neighbor already marked?
    - Yes: mark this pixel with the same value
    - No: **mark this pixel with a new value**

Equivalence LUT					
1	2				
1	2				

Remove border BLOBs – Two-pass algorithm

		2					
	2					3	
					1	1	
	1		1			1	
	1	1	1			1	
	1						
	1				1	1	
	2		2	2	2	2	

Equivalence LUT					
1	2	3			
1	2	3			

- Mark border object pixels
- Update equivalence LUT
- Loop entire image
  - Is it an object pixel?
  - Is a neighbor already marked?
    - Yes: mark this pixel with the same value
    - No: mark this pixel with a new value
      Add to equivalence LUT

Remove border BLOBs – Two-pass algorithm

		2					
	2					3	
					3	1	
	1		1			1	
	1	1	1			1	
	1						
	1				1	1	
	2		2	2	2	2	

- Mark border object pixels
- Update equivalence LUT
- Loop entire image
  - Is it an object pixel?
  - Is a neighbor already marked?
    - Yes: **mark this pixel with the same value**
    - No: mark this pixel with a new value
      Add to equivalence LUT

Equivalence LUT					
1	2	3			
1	2	3			

Remove border BLOBs – Two-pass algorithm

		2					
	2					3	
					3	3	
	1		1			1	
	1	1	1			1	
	1						
	1				1	1	
	2		2	2	2	2	

- Mark border object pixels
- Update equivalence LUT
- Loop entire image
  - Is it an object pixel?
  - Is a neighbor already marked?
    - Yes: **mark this pixel with the same value**
    - No: mark this pixel with a new value
      Add to equivalence LUT

Equivalence LUT					
1	2	3			
1	2	3			

Remove border BLOBs – Two-pass algorithm

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

- Mark border object pixels
- Update equivalence LUT
- Loop entire image
  - Is it an object pixel?
  - Is a neighbor already marked?
    - Yes: mark this pixel with the same value
    - No: **mark this pixel with a new value**
      Add to equivalence LUT

Equivalence LUT					
1	2	3	4		
1	2	3	4		

Remove border BLOBs – Two-pass algorithm

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

- Mark border object pixels
- Update equivalence LUT
- Loop entire image
  - Is it an object pixel?
  - Is a neighbor already marked?
    - Yes: mark this pixel with the same value
    - No: **mark this pixel with a new value**
      Add to equivalence LUT

Equivalence LUT					
1	2	3	4	5	
1	2	3	4	5	

Remove border BLOBs – Two-pass algorithm

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

- Mark border object pixels
- Update equivalence LUT
- Loop entire image
  - Is it an object pixel?
  - Is a neighbor already marked?
    - Yes: **mark this pixel with the same value**
    - No: mark this pixel with a new value
      Add to equivalence LUT

Equivalence LUT					
1	**2**	**3**	**4**	**5**	
1	**2**	**3**	**4**	**5**	

Remove border BLOBs – Two-pass algorithm

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

- Mark border object pixels
- Update equivalence LUT
- Loop entire image
  - Is it an object pixel?
  - Is a neighbor already marked?
    - Yes: **mark this pixel with the same value**
    - No: mark this pixel with a new value
      Add to equivalence LUT

Equivalence LUT					
1	2	3	4	5	
1	2	3	4	5	

Remove border BLOBs – Two-pass algorithm

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

- Mark border object pixels
- Update equivalence LUT
- Loop entire image
  - Is it an object pixel?
  - Is a neighbor already marked?
    - Yes: **mark this pixel with the same value**
    - No: mark this pixel with a new value
      Add to equivalence LUT

Equivalence LUT					
1	2	3	4	5	
1	2	3	4	5	

HAN_UNIVERSITY OF APPLIED SCIENCES

Remove border BLOBs – Two-pass algorithm

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

Equivalence LUT					
1	2	3	4	5	
1	2	3	4	4	

- Mark border object pixels
- Update equivalence LUT
- Loop entire image
  - Is it an object pixel?
  - Is a neighbor already marked?
    - Yes: mark this pixel with the same value
and
Update the LUT to the lowest neighbor value
    - No: mark this pixel with a new value
Add to equivalence LUT

Remove border BLOBs – Two-pass algorithm

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

Equivalence LUT					
1	2	3	4	5	
1	2	3	4	4	

- Mark border object pixels
- Update equivalence LUT
- Loop entire image
  - Is it an object pixel?
  - Is a neighbor already marked?
    - Yes: **mark this pixel with the same value**
      and
      Update the LUT to the lowest neighbor value
    - No: mark this pixel with a new value
      Add to equivalence LUT

Remove border BLOBs – Two-pass algorithm

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

Equivalence LUT					
1	2	3	4	5	
1	2	3	4	4	

- Mark border object pixels
- Update equivalence LUT
- Loop entire image
  - Is it an object pixel?
  - Is a neighbor already marked?
    - Yes: **mark this pixel with the same value**
      and
      Update the LUT to the lowest neighbor value
    - No: mark this pixel with a new value
      Add to equivalence LUT

Remove border BLOBs – Two-pass algorithm

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

Equivalence LUT					
1	2	3	4	5	
1	2	3	4	4	

- Mark border object pixels
- Update equivalence LUT
- Loop entire image
  - Is it an object pixel?
  - Is a neighbor already marked?
    - Yes: **mark this pixel with the same value**
      and
      Update the LUT to the lowest neighbor value
    - No: mark this pixel with a new value
      Add to equivalence LUT

Remove border BLOBs – Two-pass algorithm

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

Equivalence LUT					
1	2	3	4	5	
1	2	3	4	4	

- Mark border object pixels
- Update equivalence LUT
- Loop entire image
    - Is it an object pixel?
    - Is a neighbor already marked?
        - Yes: **mark this pixel with the same value**
and
Update the LUT to the lowest neighbor value
        - No: mark this pixel with a new value
Add to equivalence LUT

Remove border BLOBs – Two-pass algorithm

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

Equivalence LUT					
1	2	3	4	5	
1	2	3	2	4	

- Mark border object pixels
- Update equivalence LUT
- Loop entire image
  - Is it an object pixel?
  - Is a neighbor already marked?
    - Yes: mark this pixel with the same value
      and
      Update the LUT to the lowest neighbor value
    - No: mark this pixel with a new value
      Add to equivalence LUT

Remove border BLOBs – Two-pass algorithm

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

Equivalence LUT					
1	2	3	4	5	
1	2	3	2	2	

- Mark border object pixels
- Update equivalence LUT
- Loop entire image
  - Is it an object pixel?
  - Is a neighbor already marked?
    - Yes: mark this pixel with the same value
      and
      Update the LUT to the lowest neighbor value
      and
      Search the LUT for other equivalence updates
    - No: mark this pixel with a new value
      Add to equivalence LUT

Remove border BLOBs – Two-pass algorithm

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

Equivalence LUT					
1	2	3	4	5	
1	2	3	2	2	

- Mark border object pixels
- Update equivalence LUT
- Loop entire image
  - Is it an object pixel?
  - Is a neighbor already marked?
    - Yes: **mark this pixel with the same value**
      and
      Update the LUT to the lowest neighbor value
      and
      Search the LUT for other equivalence updates
    - No: mark this pixel with a new value
      Add to equivalence LUT

HAN_UNIVERSITY OF APPLIED SCIENCES

Remove border BLOBs – Two-pass algorithm

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

Equivalence LUT					
1	2	3	4	5	
1	2	3	2	2	

- Mark border object pixels
- Update equivalence LUT
- Loop entire image
  - Is it an object pixel?
  - Is a neighbor already marked?
    - Yes: **mark this pixel with the same value**
      and
      Update the LUT to the lowest neighbor value
      and
      Search the LUT for other equivalence updates
    - No: mark this pixel with a new value
      Add to equivalence LUT

HAN_UNIVERSITY OF APPLIED SCIENCES

Remove border BLOBs – Two-pass algorithm

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

- Loop entire image

Equivalence LUT					
1	2	3	4	5	
1	2	3	2	2	

Remove border BLOBs – Two-pass algorithm

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

- Loop entire image
  - Is it an object pixel?
    - Is the corresponding LUT value set to 2?
      - Yes: **assign background (0)**

Equivalence LUT					
1	2	3	4	5	
1	2	3	2	2	

Remove border BLOBs – Two-pass algorithm

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

- Loop entire image
  - Is it an object pixel?
    - Is the corresponding LUT value set to 2?
      - Yes: **assign background (0)**

Equivalence LUT					
1	2	3	4	5	
1	2	3	2	2	

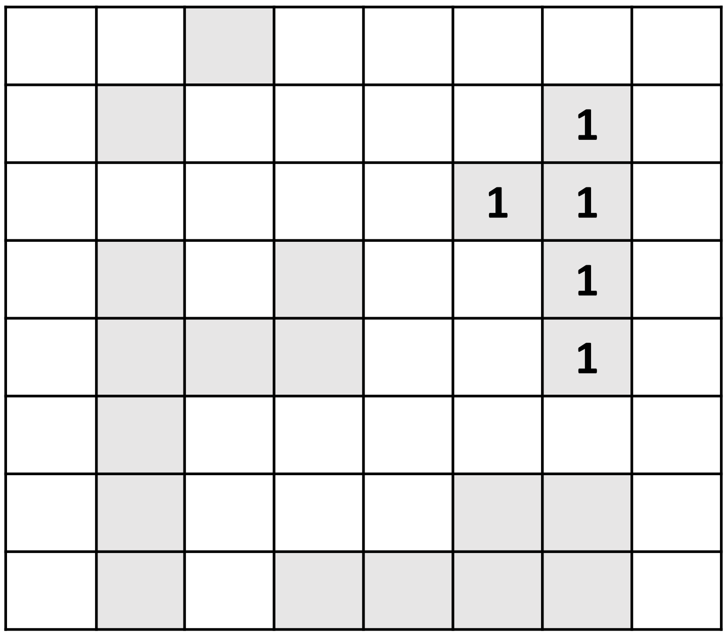
Remove border BLOBs – Two-pass algorithm

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

Equivalence LUT					
1	2	3	4	5	
1	2	3	2	2	

- Loop entire image
  - Is it an object pixel?
    - Is the corresponding LUT value set to 2?
      - Yes: assign background (0)
      - No: **assign foreground (1)**

Remove border BLOBs – Two-pass algorithm



- Loop entire image
  - Is it an object pixel?
    - Is the corresponding LUT value set to 2?
      - Yes: assign background (0)
      - No: **assign foreground (1)**

Equivalence LUT					
1	**2**	**3**	**4**	**5**	
1	**2**	**3**	**2**	**2**	

Remove border BLOBs – Two-pass algorithm

						1	
					1	1	
						1	
						1	

- The size of the equivalence LUT decides the number of labels that can be used
- The number of required labels is application depended, so we let the application decide by providing an argument

Equivalence LUT					
1	**2**	**3**	**4**	**5**	
1	**2**	**3**	**2**	**2**	

Remove border BLOBs – Two-pass algorithm

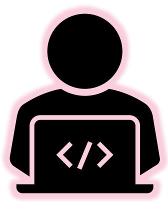
uint32_t **removeBorderBlobsTwoPass**(
const image_t ***src**, image_t ***dst,**
const eConnected **connected**, const uint32_t **lutSize**);

See file **EVDK_Operators\morphological_filters.c**

```
// Threshold the image
threshold2Means(src, tmp, BRIGHTNESS_DARK);

// Remove the border BLOBs
removeBorderBlobsTwoPass(tmp, dst, CONNECTED_EIGHT, 128);
```

EVD1 – Assignment



Study guide

Week 5

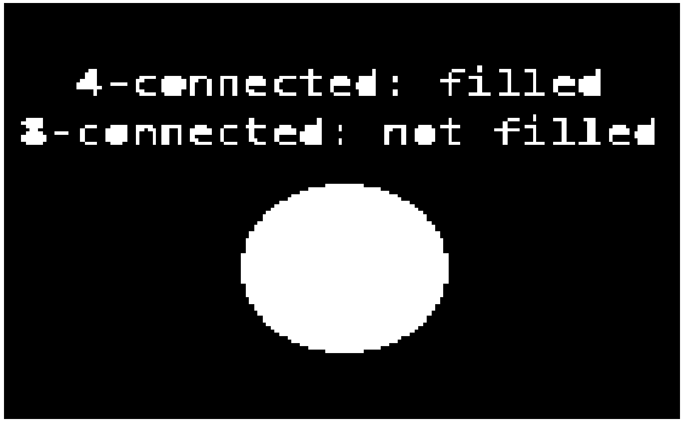
1 Morphological filters – removeBorderBlobsTwoPass()

HAN_UNIVERSITY OF APPLIED SCIENCES

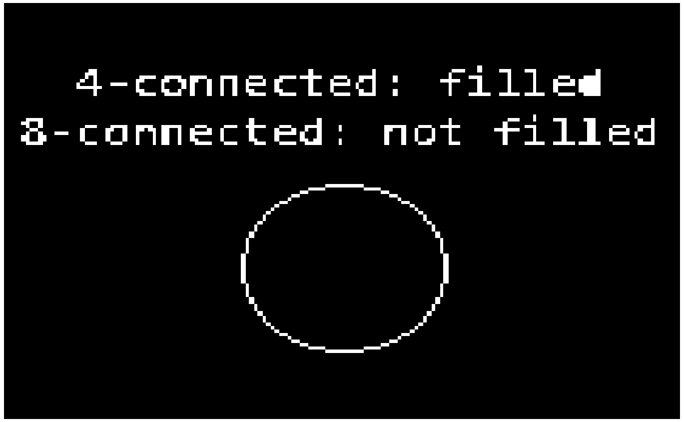
Fill holes

- Fills the 4/8-connected holes in binary objects

Fill holes - example



4-connected



8-connected

HAN_UNIVERSITY OF APPLIED SCIENCES

Fill holes

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

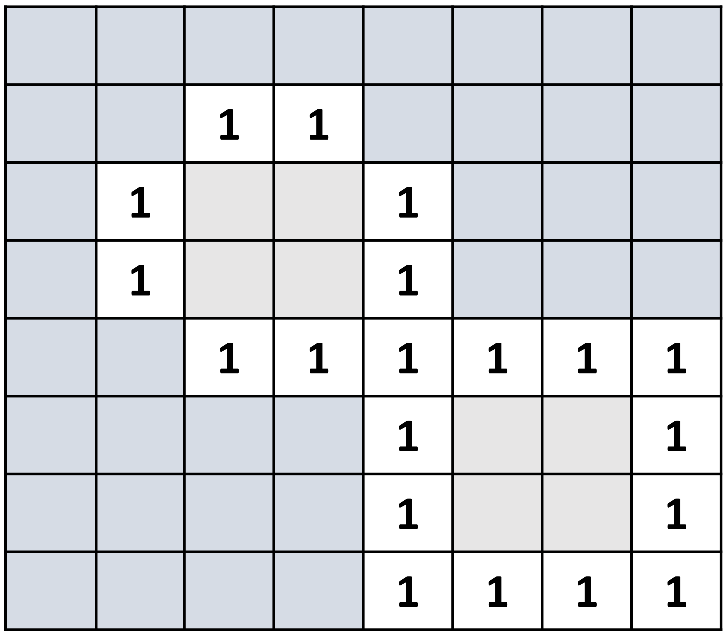
- Where to start?

Fill holes

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

- Where to start?
- However, the algorithm is very similar to removing border BLOBs, if:
  - we define a hole as not being connected to the background

Fill holes



- Where to start?
- However, the algorithm is very similar to removing border BLOBs, if:
  - we define a hole as not being connected to the background
  - And the background has all 0 pixels connected to the border of the image

Fill holes

2	2	2	2	2	2	2	2
2		1	1				2
2	1			1			2
2	1			1			2
2		1	1	1	1	1	1
2				1			1
2				1			1
2	2	2	2	1	1	1	1

- Mark background border pixels

Fill holes

2	2	2	2	2	2	2	2
2	2	1	1	2	2	2	2
2	1			1	2	2	2
2	1			1	2	2	2
2	2	1	1	1	1	1	1
2	2	2	2	1			1
2	2	2	2	1			1
2	2	2	2	1	1	1	1

- Mark background border pixels
- Mark all adjacent pixels

Fill holes

2	2	2	2	2	2	2	2
2	2	1	1	2	2	2	2
2	1	1	1	1	2	2	2
2	1	1	1	1	2	2	2
2	2	1	1	1	1	1	1
2	2	2	2	1	1	1	1
2	2	2	2	1	1	1	1
2	2	2	2	1	1	1	1

- Mark background border pixels
- Mark all adjacent pixels
- Set background pixels to foreground pixels

Fill holes

		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	1

- Mark background border pixels
- Mark all adjacent pixels
- Set background pixels to foreground
- Set marked pixels to background

Fill holes

		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	1

- Two implementations
  - Iterative algorithm
  - Two-pass algorithm

Fill holes – Iterative algorithm

uint32_t **fillHolesIterative**(const image_t ***src**, image_t ***dst,**
const eConnected **connected**);

See file **EVDK_Operators\morphological_filters.c**

```
// Threshold the image
threshold2Means(src, tmp, BRIGHTNESS_DARK);

// Remove the border BLOBs
fillHolesIterative(tmp, dst, CONNECTED_EIGHT);
```

Fill holes – Two-pass algorithm

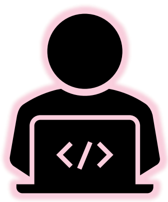
uint32_t **fillHolesTwoPass**(const image_t ***src**, image_t ***dst,**
const eConnected **connected** ,
const uint32_t **lutSize**);

See file **EVDK_Operators\morphological_filters.c**

```
// Threshold the image
threshold2Means(src, tmp, BRIGHTNESS_DARK);

// Remove the border BLOBs
fillHolesTwoPass(tmp, dst, CONNECTED_EIGHT, 128);
```

EVD1 – Assignment



Study guide

Week 5

2 Morphological filters – fillHolesTwoPass()

Dilation and erosion

- Dilation of an object increases its geometrical area
- Dilation is defined as the union of all vector additions of all pixels a in object A with all pixels b in the structuring function B:

$$A \oplus B = \{t \in Z^2 : t = a + b, a \in A, b \in B\}$$

where
t is an element of the image space Z^2

- Erosion of an object decreases its geometrical area
- Erosion is defined as the complement of the resulting dilation of the complement of object A with structuring function B:

$$A \ominus B = (A^c \oplus B)^c$$

Closing and opening

- All other morphological filters are derived from dilation and erosion

- Closing
- Reduces inward bumps and (small) holes
- Is defined as the dilation followed by an erosion of the dilated object

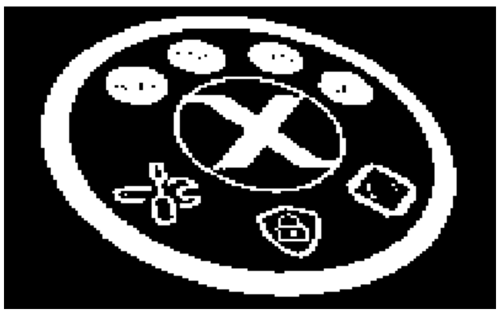
$$close(A, B) = (A \oplus B) \ominus B$$

- Opening
- Reduces outward bumps
- Is defined as the erosion followed by a dilation of the eroded object

$$open(A, B) = (A \ominus B) \oplus B$$

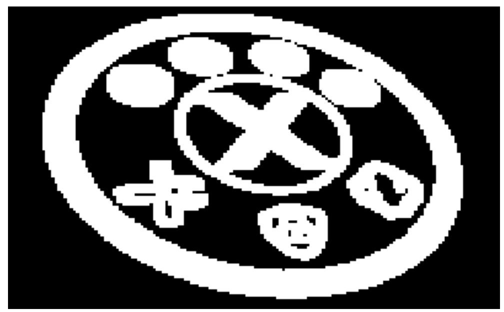
Examples

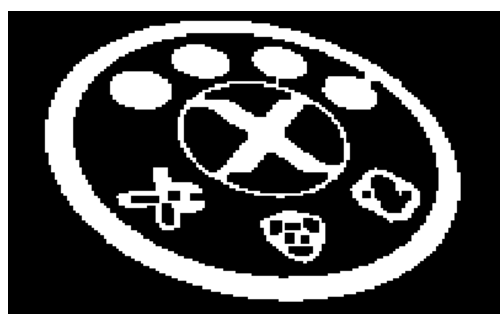
A

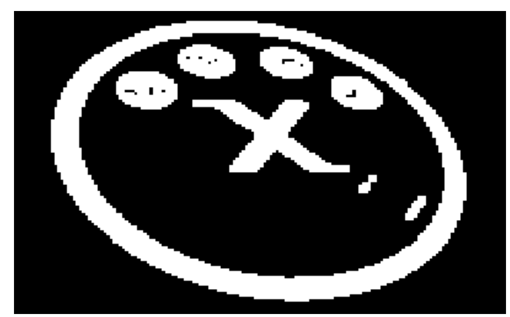


B

1	1	1
1	1	1
1	1	1

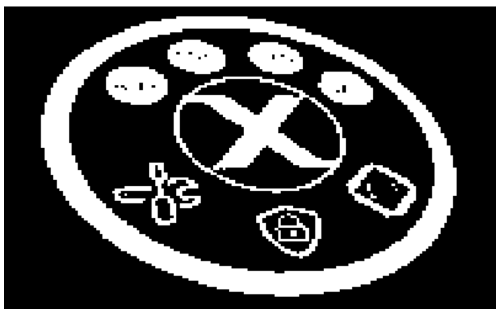






Examples

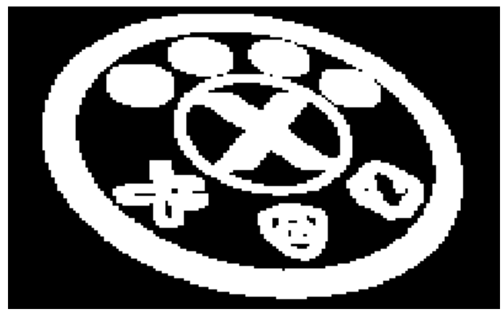
A



B

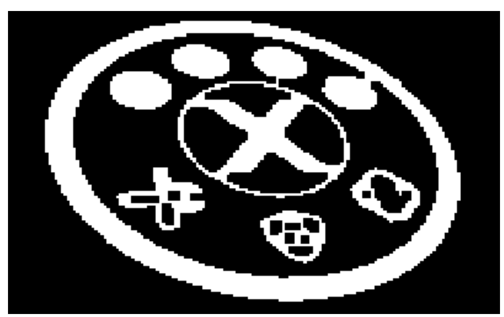
1	1	1
1	1	1
1	1	1

Dilation

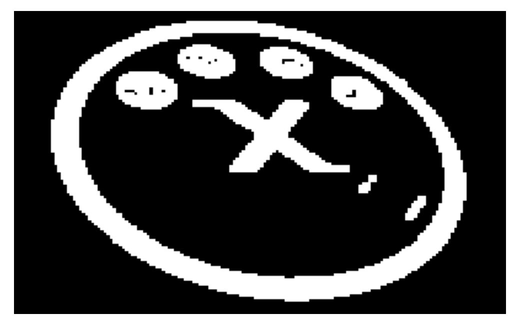


Erosion

Closing

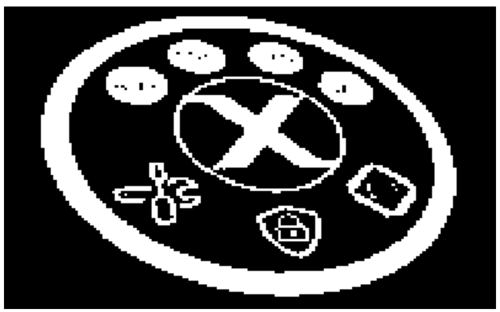


Opening



Examples

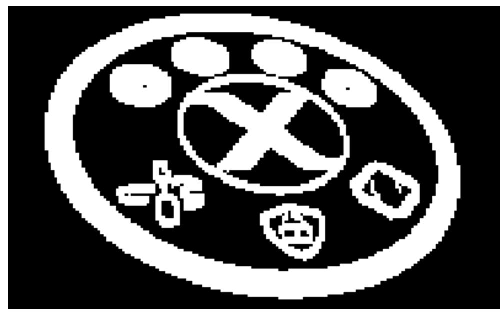
A



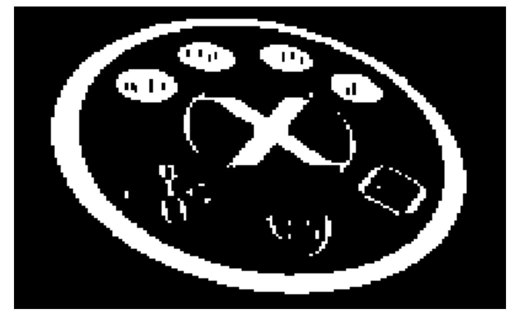
B

0	1	0
0	1	0
0	1	0

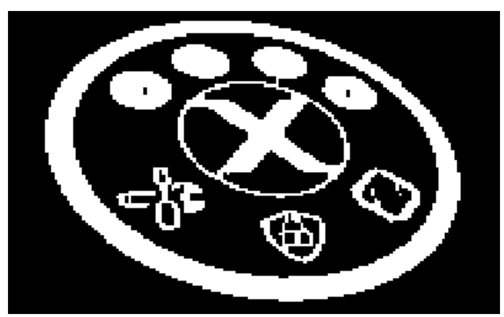
Dilation



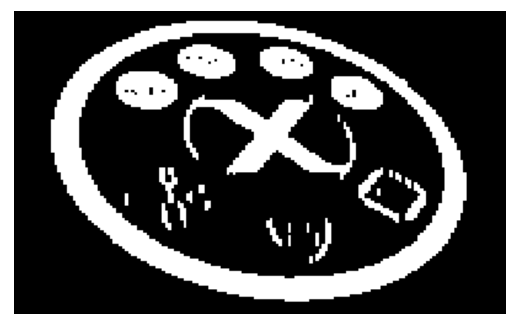
Erosion



Closing



Opening



Binary skeleton

- Defines a unique compressed geometrical representation of an object
- Does not necessarily produce a fully connected object
- Is defined as the union of the set of pixels computed from the difference of the $n-1_{th}$ eroded image and the opening of the n_{th} eroded image:

$$K_n(A) = erode_{n-1}(A) - open(erode_n(A), B)$$

where

$erode_n(A) = A \ominus B_n$ is the n_{th} erosion of the original image A with structuring function B

Binary skeleton

- The skeleton image is then given by the union of all $K_n(A)$ over all erosions
- The number of erosions n required by the skeleton algorithm is the number of erosions of the original image A by the structuring function B that yields the null image:

$$erode_n(A) = A \ominus B_n = \emptyset$$

HAN_UNIVERSITY OF APPLIED SCIENCES

Binary skeleton - example

B

0	1	0
1	1	1
0	1	0

$A = erode_0(A, B)$

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

Binary skeleton - example

B

0	1	0
1	1	1
0	1	0

$A = erode_0(A, B)$

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

$erode_1(A, B)$

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

Binary skeleton - example

B

0	1	0
1	1	1
0	1	0

$\overbrace{\qquad\qquad}^{open(erode_0(A,B),B)}$

$A = erode_0(A,B)$

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

$erode_1(A,B)$

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

$dilate(erode_1(A,B),B)$

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

Binary skeleton - example

$$K_1(A) = erode_0(A, B) - open(erode_0(A, B), B)$$

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

Binary skeleton - example

B

0	1	0
1	1	1
0	1	0

$erode_1(A, B)$

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

Binary skeleton - example

B

0	1	0
1	1	1
0	1	0

$erode_1(A, B)$

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

$erode_2(A, B)$

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
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

Binary skeleton - example

B

0	1	0
1	1	1
0	1	0

$open(erode_1(A,B),B)$

$erode_1(A,B)$

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

$erode_2(A,B)$

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
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

$dilate(erode_2(A,B),B)$

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	1	0	0	0	0
0	0	1	1	1	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

Binary skeleton - example

$$K_2(A) = erode_1(A, B) - open(erode_1(A, B), B)$$

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

$$K_1(A) \cup K_2(A)$$

Binary skeleton - example

B

0	1	0
1	1	1
0	1	0

$erode_2(A, B)$

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
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

Binary skeleton - example

B

0	1	0
1	1	1
0	1	0

$erode_2(A, B)$

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
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

$erode_3(A, B)$

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
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

$Done!$
$Nothing\ left\ to\ erode.$

Binary skeleton - example

B

0	1	0
1	1	1
0	1	0

$$\overbrace{\quad\quad\quad\quad}^{open(erode_2(A,B),B)}$$

$erode_2(A,B)$

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
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

$erode_3(A,B)$

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
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

$dilate(erode_3(A,B),B)$

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
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

$Done!$
$Nothing\ left\ to\ erode.$

Binary skeleton - example

$$K_3(A) = erode_2(A,B) - open(erode_2(A,B),B)$$

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

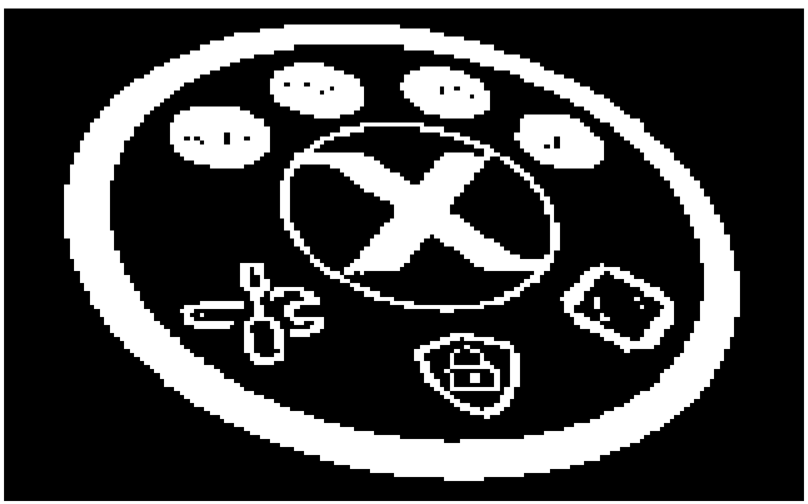
$$K_1(A) \cup K_2(A) \cup K_3(A)$$

Binary skeleton - example

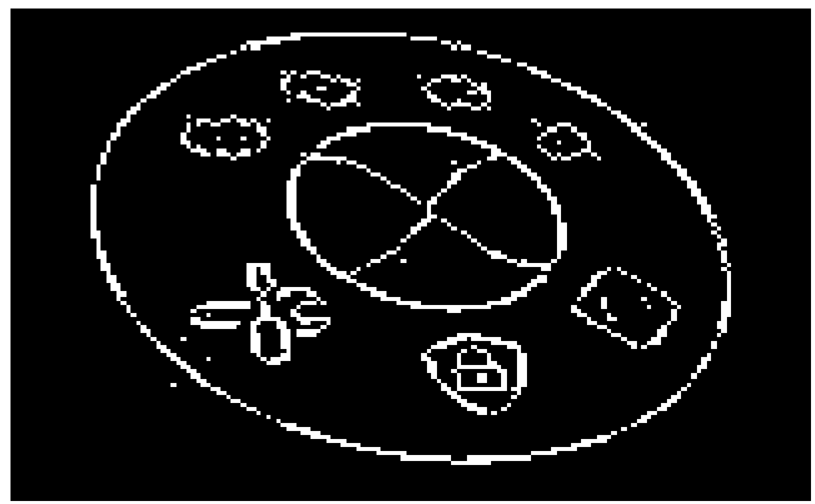
B

0	1	0
1	1	1
0	1	0

A



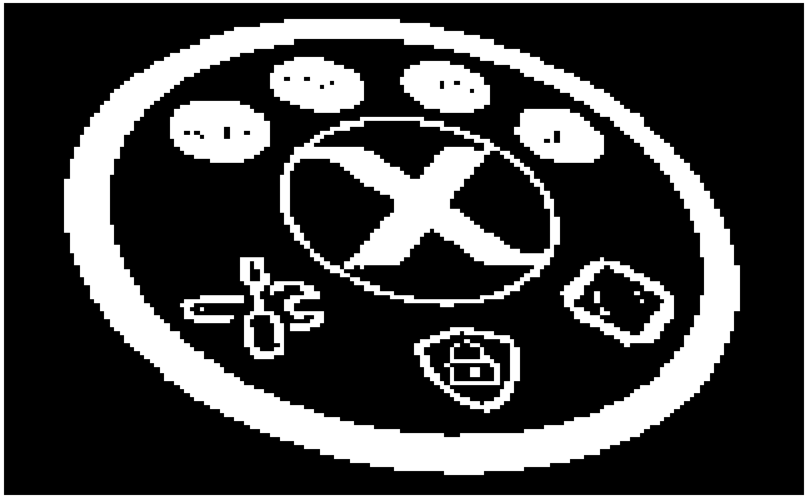
$K_n(A)$



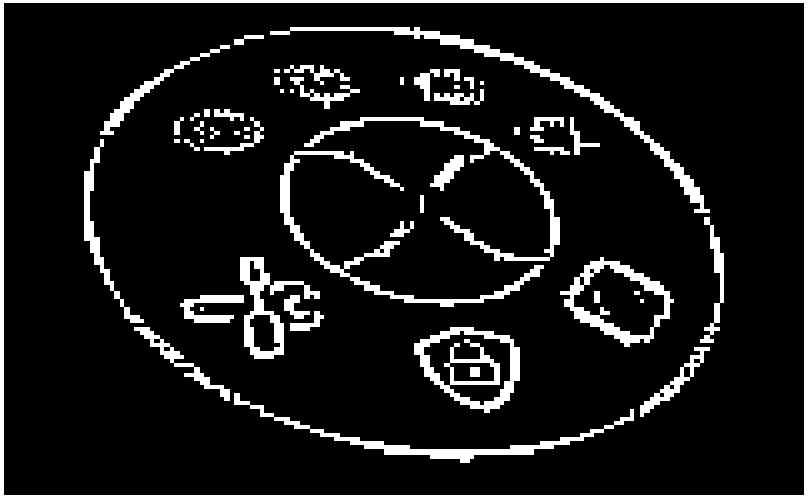
Binary skeleton - example

B	1	0	1
	0	1	0
	1	0	1

A



$K_n(A)$



Binary skeleton - algorithm

void **skeleton**(const image_t ***src**, image_t ***dst**,
const uint8_t ***mask**, const uint8_t **n**);

See file **EVDK_Operators\morphological_filters.c**

```
// Threshold the image
threshold2Means(src, tmp, BRIGHTNESS_DARK);
removeBorderBlobsTwoPass(tmp, rbb, CONNECTED_FOUR, 256);

uint8_t mask[9] =
{
    1,0,1,
    0,1,0,
    1,0,1,
};

skeleton(rbb, dst, mask, 3);
```

Binary hit-miss

- Use to find geometrical features
- Is defines as:

$$hitmiss(A, B, C) = (A \ominus B) \cap (A^c \ominus C)$$

where

B and C are structuring masks with the requirement:

$$B \cap C = \emptyset$$

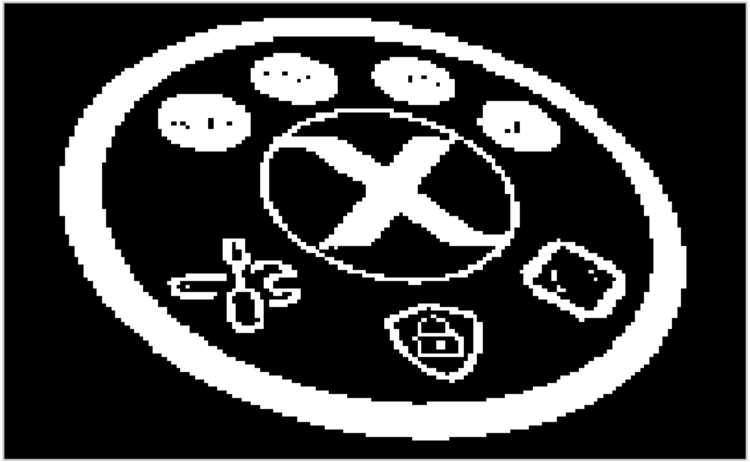
because all 1s in B are considered object pixels and all 1s in C are considered background pixels

Binary hit-miss - example

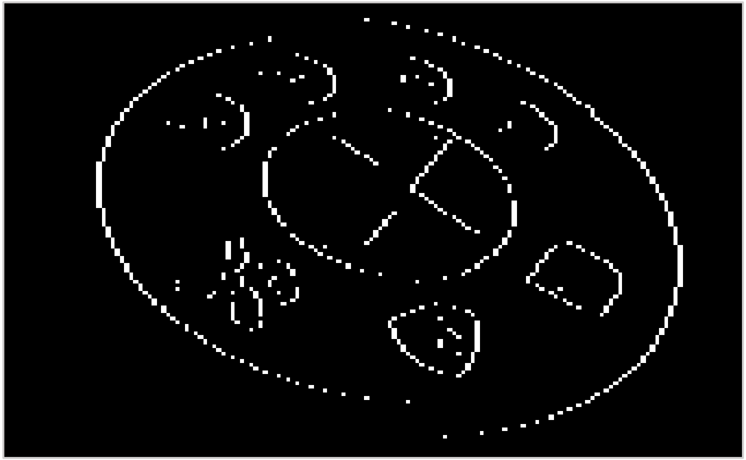
$$B = \begin{array}{|c|c|c|}\hline 0 & 0 & 0 \\ \hline 1 & 1 & 0 \\ \hline 0 & 0 & 0 \\ \hline \end{array} \qquad C = \begin{array}{|c|c|c|}\hline 0 & 0 & 0 \\ \hline 0 & 0 & 1 \\ \hline 0 & 0 & 0 \\ \hline \end{array}$$

$$\textit{Alternative representation} \quad \begin{array}{|c|c|c|}\hline - & - & - \\ \hline 1 & 1 & 0 \\ \hline - & - & - \\ \hline \end{array}$$

A



$hitmiss(A, B, C)$



Binary hit-miss - example

B

0	0	0
0	1	1
0	0	0

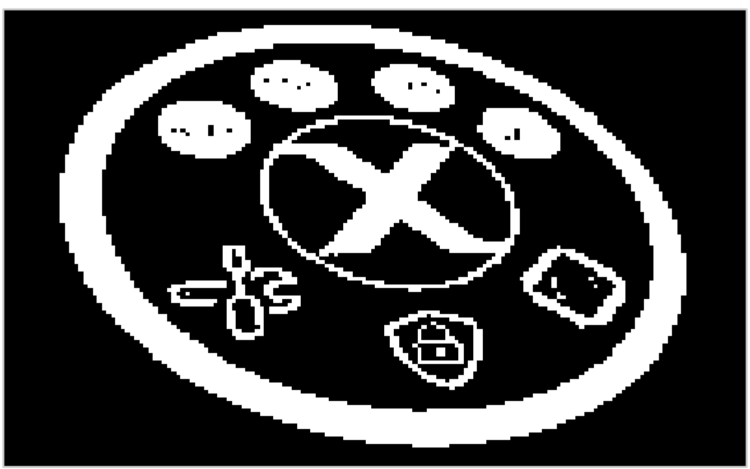
C

0	0	0
1	0	0
0	0	0

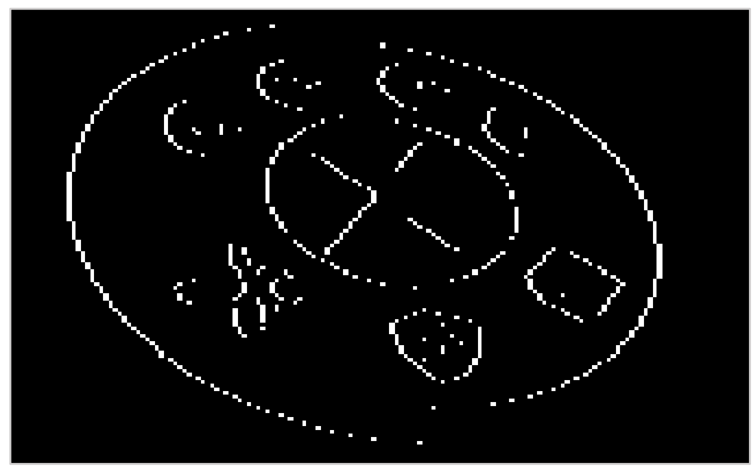
$Alternative\ representation$

-	-	-
0	1	1
-	-	-

A



$hitmiss(A, B, C)$



Binary hit-miss - algorithm

void **hitmiss**(const image_t ***src**, image_t ***dst**,
const uint8_t ***m1**, const uint8_t ***m2**);

See file **EVDK_Operators\morphological_filters.c**

Binary outline

- Change all of the object's pixels to the background value, except those pixels that lie on the object's contour
- The contour width is determined by the structuring element
- The result is the eroded image subtracted from the original image, or the original image subtracted from the dilated image
- Is defined as

$$outline(A, B) = A - (A \ominus B)$$

or

$$outline(A, B) = (A \oplus B) - A$$

where

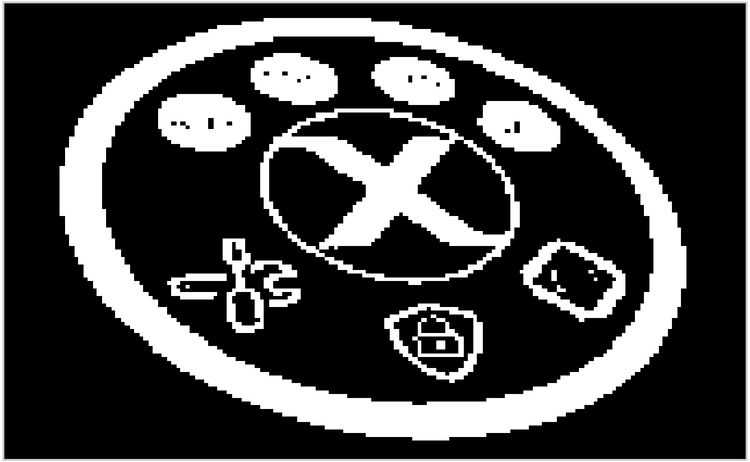
B is the structuring mask

Binary outline - example

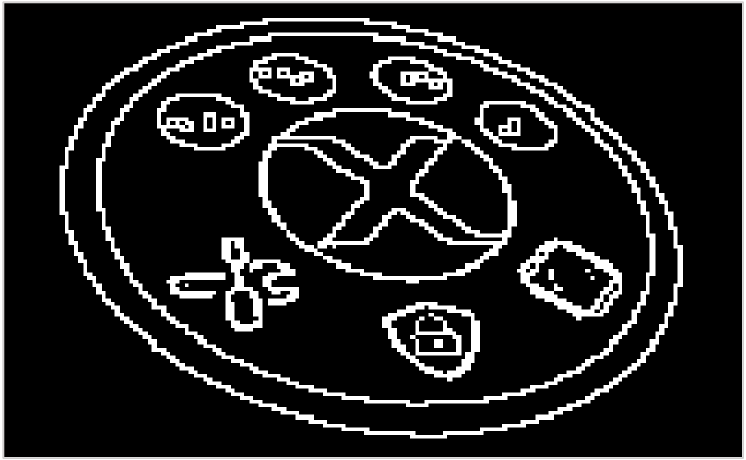
B

1	1	1
1	1	1
1	1	1

A



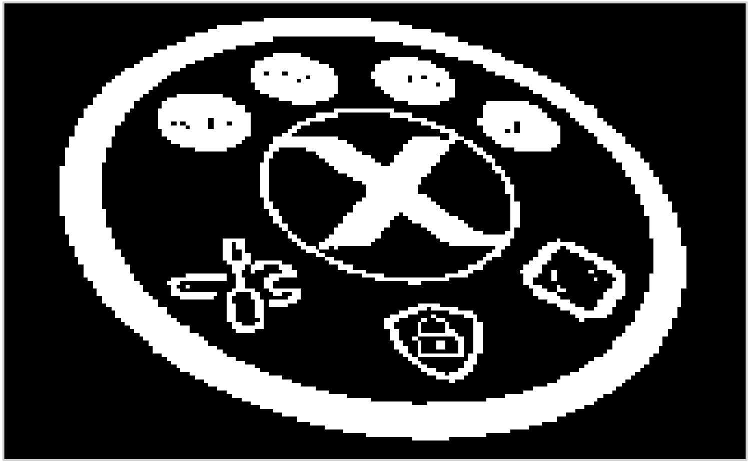
$outline(A, B) = A - (A \ominus B)$



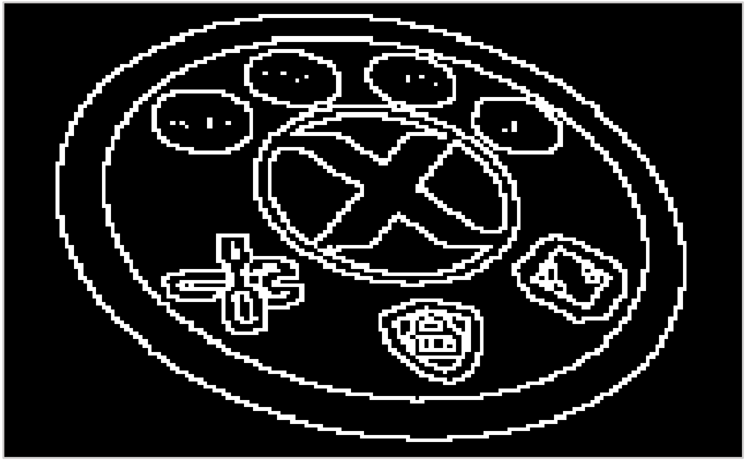
Binary outline - example

$$B = \begin{array}{|c|c|c|} \hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline \end{array}$$

A



$outline(A, B) = (A \oplus B) - A$



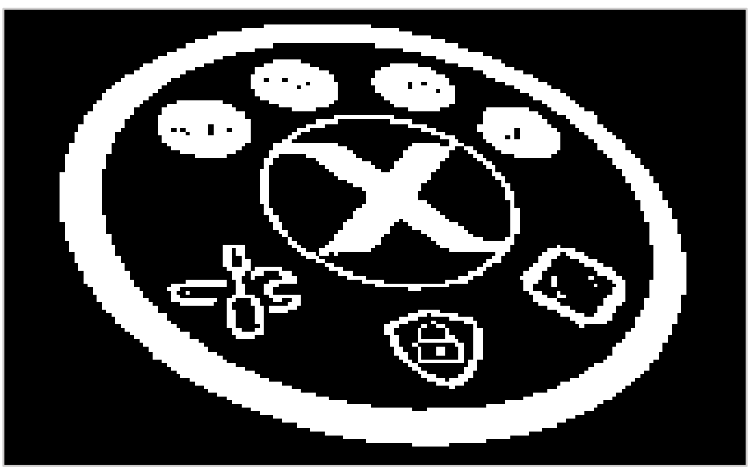
HAN_UNIVERSITY OF APPLIED SCIENCES

Binary outline - example

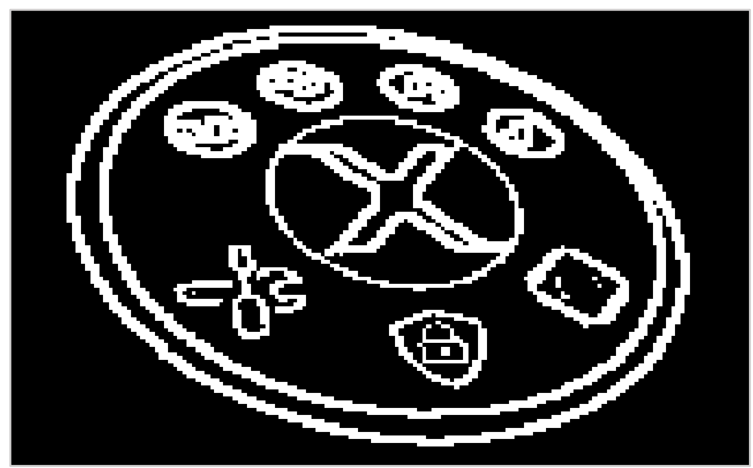
B

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

A



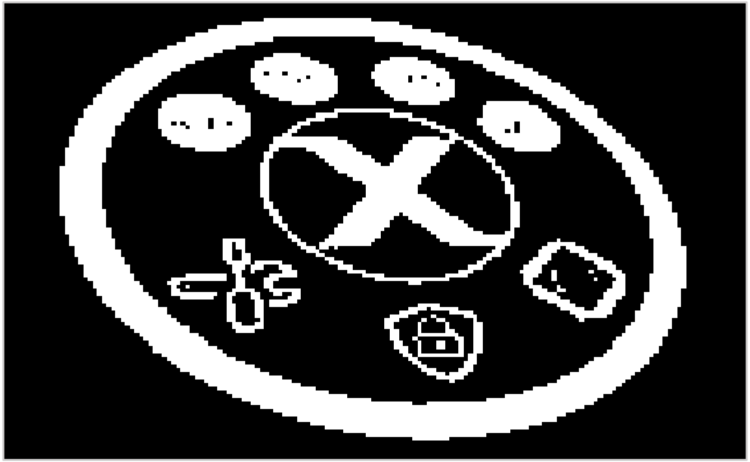
$outline(A, B)$



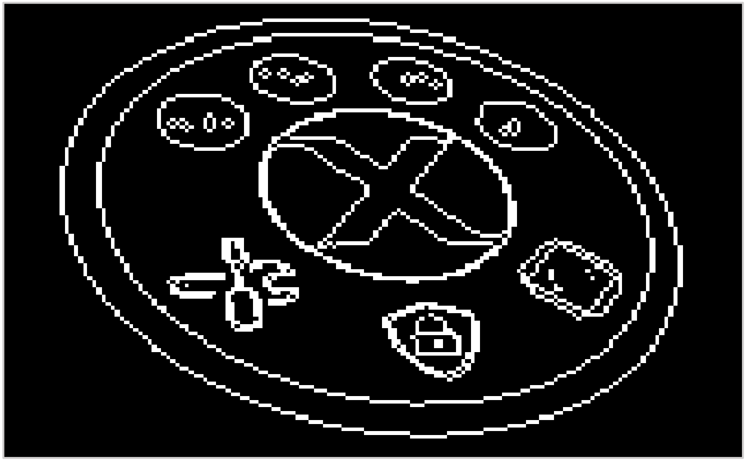
Binary outline - example

	0	1	0
B	1	1	1
	0	1	0

A



$outline(A, B)$



Binary outline - algorithm

```
void outline( const image_t *src, image_t *dst,
              const uint8_t *m, const uint8_t n);
```

See file **EVDK_Operators\morphological_filters.c**

HAN_UNIVERSITY OF APPLIED SCIENCES

References

- Myler, H. R., & Weeks, A. R. (2009). *The pocket handbook of image processing algorithms in C*. Prentice Hall Press.

HAN_UNIVERSITY OF APPLIED SCIENCES