HAN AEA - Embedded Vision & Machine Learning

EVD1 - Week 5

Morphological Filters

By Hugo Arends

HAN_UNIVERSITY OF APPLIED SCIENCES

Morphological filters

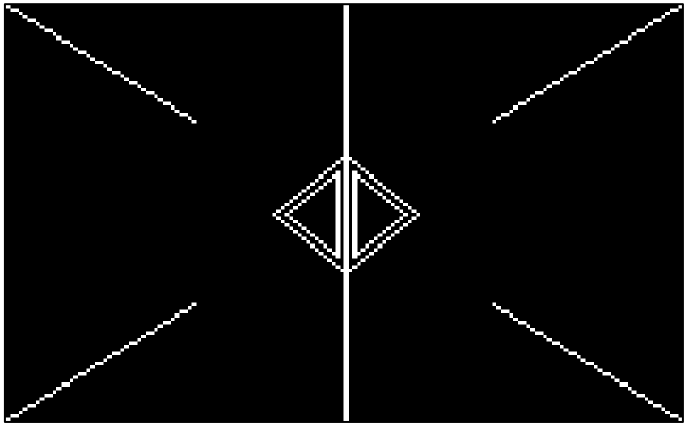
- Are used prior to pattern recognition and object classification
- Changes the geometrical shape of the objects
- The goal is to smooth the object’s contours and to decompose objects in their fundamental shapes

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

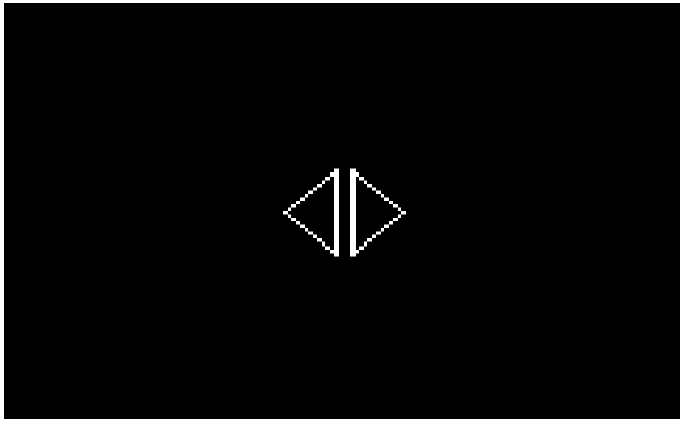
Remove border BLOBs

- 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	

- Copy src to dst

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	

- Copy src to dst
- 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	

- Copy src to dst
- Mark border object pixels
- While changes
  - Loop entire image and assign marker value if a neighbour 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	

- Copy src to dst
- Mark border object pixels
- While changes
  - Loop entire image and assign marker value if a neighbour 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	

- Copy src to dst
- Mark border object pixels
- While changes
  - Loop entire image and assign marker value if a neighbour 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	

- Copy src to dst
- Mark border object pixels
- While changes
  - Loop entire image and assign marker value if a neighbour 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	

- Copy src to dst
- Mark border object pixels
- While changes
  - Loop entire image and assign marker value if a neighbour 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	

- Copy src to dst
- Mark border object pixels
- While changes
    - Loop entire image and assign marker value if a neighbour 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	

- Copy src to dst
- Mark border object pixels
- While changes
  - Loop entire image and assign marker value if a neighbour 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	

- Copy src to dst
- Mark border object pixels
- While changes
  - Loop entire image and assign marker value if a neighbour 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	

- Copy src to dst
- Mark border object pixels
- While changes
  - Loop entire image and assign marker value if a neighbour 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	

- Copy src to dst
- Mark border object pixels
- While changes
  - Loop entire image and assign marker value if a neighbour is also marked

Remove border BLOBs – Iterative algorithm

						1	
					1	1	
						1	
						1	

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

Remove border BLOBs – Iterative algorithm

Advantage

- Easy implementation

Disadvantage

- Very slow, especially if we pass through the image in a single direction and the objects happen to 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**	

1. Initialize lookup table (lut)
2. Mark the border pixels in the destination
3. Pass 1: Label the objects, skipping the borders, and record equivalences in lut
4. Record border equivalences
5. Resolve equivalences
6. Pass 2: Assign result by using lut
7. Cleanup lut

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	

1. Initialize lookup table (lut)

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	

1. Initialize lookup table (lut)
   - By allocating memory

Equivalence LUT								
0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**...**

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	

1. Initialize lookup table (lut)
   - By allocating memory
   - The lut will record equivalences. In other words: it records label values that are connected. It is a one-dimensional array.

Equivalence LUT								
0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**...**

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	

1. Initialize lookup table (lut)
   - By allocating memory
   - The lut will record equivalences. In other words: it records label values that are connected. It is a one-dimensional array.
   - The first three labels are reserved:
     - 0: Background
     - 1: Objects
     - 2: Border marker

Equivalence LUT								
0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**...**
0	**1**	**2**						

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								
0	1	2	3	4	5	6	7	...
0	1	2	0					

1. Initialize lookup table (lut)
   - By allocating memory
   - The lut will record equivalences. In other words: it records label values that are connected. It is a one-dimensional array.
   - The first three labels are reserved:
     - 0: Background
     - 1: Objects
     - 2: Border marker
   - Mark end of the lut

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	

2. Mark the border pixels in the destination

Equivalence LUT								
0	1	2	3	4	5	6	7	…
0	1	2	0					

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	

		2					
	2		2	2	2	2	

2. Mark the border pixels in the destination
   - Border pixels must be removed, so use label value 2 to mark them.
   - All other border pixels are set to the background label value 0

Equivalence LUT								
0	1	2	3	4	5	6	7	...
0	1	2	0					

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	

		2					
	2		2	2	2	2	

3. Pass 1: Label the objects, skipping the borders, and record equivalences in lut

Equivalence LUT								
0	1	2	3	4	5	6	7	...
0	1	2	0					

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	

		2					
	2		2	2	2	2	

3. Pass 1: Label the objects, skipping the borders, and record equivalences in lut
   - Loop all inner pixels

Equivalence LUT								
0	1	2	3	4	5	6	7	...
0	1	2	0					

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	

		2					
	2		2	2	2	2	

3. Pass 1: Label the objects, skipping the borders, and record equivalences in lut
   - Loop all inner pixels
   - Is it an object pixel?

Equivalence LUT								
0	1	2	3	4	5	6	7	...
0	1	2	0					

HAN_UNIVERSITY OF APPLIED SCIENCES

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	

		2					
	2		2	2	2	2	

3. Pass 1: Label the objects, skipping the borders, and record equivalences in lut
   - Loop all inner pixels
   - Is it an object pixel?
     - Yes: Is one of its neighbours labeled?

How many neighbours should be considered?

Equivalence LUT								
0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**…**
0	**1**	**2**	**0**					

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	

		2					
	2		2	2	2	2	

3. Pass 1: Label the objects, skipping the borders, and record equivalences in lut
   - Loop all inner pixels
   - Is it an object pixel?
     - Yes: Is one of its neighbours labeled?

Equivalence LUT								
0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**...**
0	**1**	**2**	**0**					

HAN_UNIVERSITY OF APPLIED SCIENCES

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	

		2					
	2						
	2		2	2	2	2	

3. Pass 1: Label the objects, skipping the borders, and record equivalences in lut
   - Loop all inner pixels
   - Is it an object pixel?
     - Yes: Is one of its neighbours labeled?
       - Yes: **Label this pixel with the same value**

Equivalence LUT								
0	1	2	3	4	5	6	7	…
0	1	2	0					

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	

		2					
	2						
	2		2	2	2	2	

3. Pass 1: Label the objects, skipping the borders, and record equivalences in lut
   - Loop all inner pixels
   - Is it an object pixel?
     - Yes: Is one of its neighbours labeled?
       - Yes: Label this pixel with the same value
     - No: **Label this pixel with background value**

Equivalence LUT								
0	1	2	3	4	5	6	7	...
0	1	2	0					

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	

		2					
	2					3	
	2		2	2	2	2	

3. Pass 1: Label the objects, skipping the borders, and record equivalences in lut
   - Loop all inner pixels
   - Is it an object pixel?
     - Yes: Is one of its neighbours labeled?
       - Yes: Label this pixel with the same value
       - No: **Assign next label value**
     - No: Label this pixel with background value

Equivalence LUT								
0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**...**
0	**1**	**2**	**0**					

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	

		2					
	2					3	
	2		2	2	2	2	

3. Pass 1: Label the objects, skipping the borders, and record equivalences in lut
   - Loop all inner pixels
   - Is it an object pixel?
     - Yes: Is one of its neighbours labeled?
       - Yes: Label this pixel with the same value
       - No: Assign next label value and **record equivalence in lut**
     - No: Label this pixel with background value

Equivalence LUT								
0	1	2	3	4	5	6	7	...
0	1	2	3	0				

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	

		2					
	2					3	
					3		
	2		2	2	2	2	

Equivalence LUT								
0	1	2	3	4	5	6	7	…
0	1	2	3	0				

3. Pass 1: Label the objects, skipping the borders, and record equivalences in lut
   - Loop all inner pixels
   - Is it an object pixel?
     - Yes: Is one of its neighbours labeled?
       - Yes: **Label this pixel with the same value**
       - No: Assign next label value and record equivalence in lut
     - No: Label this pixel with background value

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	

		2					
	2					3	
					3	3	
	2		2	2	2	2	

3. Pass 1: Label the objects, skipping the borders, and record equivalences in lut
   - Loop all inner pixels
   - Is it an object pixel?
     - Yes: Is one of its neighbours labeled?
       - Yes: **Label this pixel with the same value**
       - No: Assign next label value and record equivalence in lut
     - No: Label this pixel with background value

Equivalence LUT								
0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**...**
0	**1**	**2**	**3**	**0**				

HAN_UNIVERSITY OF APPLIED SCIENCES

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	

		2					
	2					3	
					3	3	
	4						
	2		2	2	2	2	

Equivalence LUT								
0	1	2	3	4	5	6	7	...
0	1	2	3	4	0			

3. Pass 1: Label the objects, skipping the borders, and record equivalences in lut
   - Loop all inner pixels
   - Is it an object pixel?
     - Yes: Is one of its neighbours labeled?
       - Yes: Label this pixel with the same value
       - No: **Assign next label value and record equivalence in lut**
     - No: Label this pixel with background value

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	

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

Equivalence LUT								
0	1	2	3	4	5	6	7	...
0	1	2	3	4	5	0		

3. Pass 1: Label the objects, skipping the borders, and record equivalences in lut
   - Loop all inner pixels
   - Is it an object pixel?
     - Yes: Is one of its neighbours labeled?
       - Yes: Label this pixel with the same value
       - No: **Assign next label value and record equivalence in lut**
     - No: Label this pixel with background value

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	

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

3. Pass 1: Label the objects, skipping the borders, and record equivalences in lut
   - Loop all inner pixels
   - Is it an object pixel?
     - Yes: Is one of its neighbours labeled?
       - Yes: **Label this pixel with the same value**
       - No: Assign next label value and record equivalence in lut
     - No: Label this pixel with background value

Equivalence LUT								
0	1	2	3	4	5	6	7	...
0	1	2	3	4	5	0		

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	

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

Equivalence LUT								
0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**...**
0	**1**	**2**	**3**	**4**	**5**	**0**		

3. Pass 1: Label the objects, skipping the borders, and record equivalences in lut
   - Loop all inner pixels
   - Is it an object pixel?
     - Yes: Is one of its neighbours labeled?
       - Yes: **Label this pixel with the same value**
       - No: Assign next label value and record equivalence in lut
     - No: Label this pixel with background value

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	

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

3. Pass 1: Label the objects, skipping the borders, and record equivalences in lut
   - Loop all inner pixels
   - Is it an object pixel?
     - Yes: Is one of its neighbours labeled?
       - Yes: **Label this pixel with the <u>lowest</u> value**
       - No: Assign next label value and record equivalence in lut
     - No: Label this pixel with background value

Equivalence LUT								
0	1	2	3	4	5	6	7	...
0	1	2	3	4	5	0		

HAN_UNIVERSITY OF APPLIED SCIENCES

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	

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

3. Pass 1: Label the objects, skipping the borders, and record equivalences in lut
   - Loop all inner pixels
   - Is it an object pixel?
     - Yes: Is one of its neighbours labeled?
       - Yes: Label this pixel with the <u>lowest</u> value and **record equivalence(s)**
       - No: Assign next label value and record equivalence in lut
     - No: Label this pixel with background value

Equivalence LUT								
0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**…**
0	**1**	**2**	**3**	**4**	**4**	**0**		

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	

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

Equivalence LUT								
0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**…**
0	**1**	**2**	**3**	**4**	**4**	**0**		

3. Pass 1: Label the objects, skipping the borders, and record equivalences in lut
   - Loop all inner pixels
   - Is it an object pixel?
     - Yes: Is one of its neighbours labeled?
       - Yes: **Label this pixel with the <u>lowest</u> value** and **record equivalence(s)**
       - No: Assign next label value and record equivalence in lut
     - No: Label this pixel with background value

HAN_UNIVERSITY OF APPLIED SCIENCES

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	

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

Equivalence LUT								
0	1	2	3	4	5	6	7	...
0	1	2	3	4	4	0		

3. Pass 1: Label the objects, skipping the borders, and record equivalences in lut
   - Loop all inner pixels
   - Is it an object pixel?
     - Yes: Is one of its neighbours labeled?
       - Yes: **Label this pixel with the <u>lowest</u> value** and record equivalence(s)
       - No: Assign next label value and record equivalence in lut
     - No: Label this pixel with background value

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	

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

Equivalence LUT								
0	1	2	3	4	5	6	7	…
0	1	2	3	4	4	0		

3. Pass 1: Label the objects, skipping the borders, and record equivalences in lut
   - Loop all inner pixels
   - Is it an object pixel?
     - Yes: Is one of its neighbours labeled?
       - Yes: **Label this pixel with the <u>lowest</u> value** and record equivalence(s)
       - No: Assign next label value and record equivalence in lut
     - No: Label this pixel with background value

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	

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

Equivalence LUT								
0	1	2	3	4	5	6	7	…
0	1	2	3	4	4	0		

3. Pass 1: Label the objects, skipping the borders, and record equivalences in lut
   - Loop all inner pixels
   - Is it an object pixel?
     - Yes: Is one of its neighbours labeled?
       - Yes: **Label this pixel with the <u>lowest</u> value** and record equivalence(s)
       - No: Assign next label value and record equivalence in lut
     - No: Label this pixel with background value

HAN_UNIVERSITY OF APPLIED SCIENCES

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	

		2					
	2					3	
					3	3	
	4		5			3	
	4	4	4			3	
	4						
	4				6		
	2		2	2	2	2	

3. Pass 1: Label the objects, skipping the borders, and record equivalences in lut
   - Loop all inner pixels
   - Is it an object pixel?
     - Yes: Is one of its neighbours labeled?
       - Yes: Label this pixel with the <u>lowest</u> value and record equivalence(s)
       - No: **Assign next label value and record equivalence in lut**
     - No: Label this pixel with background value

Equivalence LUT								
0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**...**
0	**1**	**2**	**3**	**4**	**4**	**6**	**0**	

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	

		2					
	2					3	
					3	3	
	4		5			3	
	4	4	4			3	
	4						
	4				6	6	
	2		2	2	2	2	

Equivalence LUT								
0	1	2	3	4	5	6	7	...
0	1	2	3	4	4	6	0	

3. Pass 1: Label the objects, skipping the borders, and record equivalences in lut
   - Loop all inner pixels
   - Is it an object pixel?
     - Yes: Is one of its neighbours labeled?
       - Yes: **Label this pixel with the <u>lowest</u> value** and record equivalence(s)
       - No: Assign next label value and record equivalence in lut
     - No: Label this pixel with background value

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	

		2					
	2					3	
					3	3	
	4		5			3	
	4	4	4			3	
	4						
	4				6	6	
	2		2	2	2	2	

Equivalence LUT								
0	1	2	3	4	5	6	7	...
0	1	2	3	4	4	6	0	

3. Pass 1: Label the objects, skipping the borders, and record equivalences in lut
   - Loop all inner pixels
   - Is it an object pixel?
     - Yes: Is one of its neighbours labeled?
       - Yes: Label this pixel with the lowest value and record equivalence(s)
       - No: Assign next label value and record equivalence in lut
     - No: Label this pixel with background value

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	

		2					
	2					3	
					3	3	
	4		5			3	
	4	4	4			3	
	4						
	4				6	6	
	2		2	2	2	2	

4. Record border equivalences

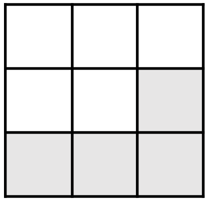
Equivalence LUT								
0	1	2	3	4	5	6	7	…
0	1	2	3	4	4	6	0	

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	

		2					
	2					3	
					3	3	
	4		5			3	
	4	4	4			3	
	4						
	4				6	6	
	2		2	2	2	2	

4. Record border equivalences
   - Must check equivalences due to skipped borders and skipped neighbours



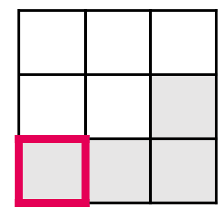
Equivalence LUT								
0	1	2	3	4	5	6	7	…
0	1	2	3	4	4	6	0	

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	

		2					
	2					3	
					3	3	
	4		5			3	
	4	4	4			3	
	4						
	4				6	6	
	2		2	2	2	2	

4. Record border equivalences
   - Must check equivalences due to skipped borders and skipped neighbours



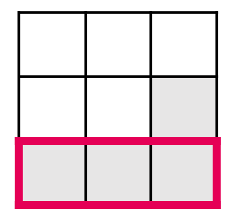
Equivalence LUT								
0	1	2	3	4	5	6	7	...
0	1	2	3	4	4	6	0	

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	

		2					
	2					3	
					3	3	
	4		5			3	
	4	4	4			3	
	4						
	4				6	6	
	2		2	2	2	2	

4. Record border equivalences
   - Must check equivalences due to skipped borders and skipped neighbours



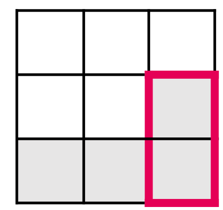
Equivalence LUT								
0	1	2	3	4	5	6	7	...
0	1	2	3	4	4	6	0	

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	

		2					
	2					3	
					3	3	
	4		5			3	
	4	4	4			3	
	4						
	4				6	6	
	2		2	2	2	2	

4. Record border equivalences
   - Must check equivalences due to skipped borders and skipped neighbours



Equivalence LUT								
0	1	2	3	4	5	6	7	...
0	1	2	3	4	4	6	0	

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	

		2					
	$\boxed{2}$					3	
					3	3	
	4		5			3	
	4	4	4			3	
	4						
	4				6	6	
	2		2	2	2	2	

4. Record border equivalences
   - Is the pixel labeled?
     - Yes: Is a border pixel marked?
       - No: **No need to update lut**

Equivalence LUT								
0	1	2	3	4	5	6	7	...
0	1	2	3	4	4	6	0	

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	

		2					
	2					3	
					3	$\boxed{3}$	
	4		5			3	
	4	4	4			3	
	4						
	4				6	6	
	2		2	2	2	2	

4. Record border equivalences
   - Is the pixel labeled?
     - Yes: Is a border pixel marked?
       - No: **No need to update lut**

Equivalence LUT								
0	1	2	3	4	5	6	7	...
0	1	2	3	4	4	6	0	

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	

		2					
	2					3	
					3	3	
	4		5			3	
	4	4	4			3	
	4						
	$\boxed{4}$				6	6	
	2		2	2	2	2	

4. Record border equivalences
   - Is the pixel labeled?
     - Yes: Is a border pixel marked?
       - No: No need to update lut
       - Yes: **Update lut**

Equivalence LUT								
0	1	2	3	4	5	6	7	...
0	1	2	3	2	4	6	0	

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	

		2					
	2					3	
					3	3	
	4		5			3	
	4	4	4			3	
	4						
	4				6	6	
	2		2	2	2	2	

4. Record border equivalences
   - Is the pixel labeled?
     - Yes: Is a border pixel marked?
       - No: No need to update lut
       - Yes: update lut and **update lut initial assigned label too**

Equivalence LUT								
0	1	2	3	4	5	6	7	...
0	1	2	3	2	4	6	0	

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	

		2					
	2					3	
					3	3	
	4		5			3	
	4	4	4			3	
	4						
	4				6	6	
	2		2	2	2	2	

4. Record border equivalences
   - Is the pixel labeled?
     - Yes: Is a border pixel marked?
       - No: No need to update lut
       - Yes: update lut and **update lut initial assigned label too**

Equival								
0	1	2	3	4				
0	1	2	3	2	4	6	0	

Why?

			1	
	1	1		
1				

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	

		2					
	2					3	
					3	3	
	4		5			3	
	4	4	4			3	
	4						
	4				6	6	
	2		2	2	2	2	

4. Record border equivalences
   - Is the pixel labeled?
     - Yes: Is a border pixel marked?
       - No: No need to update lut
       - Yes: update lut and **update lut initial assigned label too**

Equival								
0	1	2	3	4				
0	1	2	3	2	4	6	0	

Why?

			3	
	4	3		
2				

0	1	2	3	4
0	1	2	3	3

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	

		2					
	2					3	
					3	3	
	4		5			3	
	4	4	4			3	
	4						
	4				6	6	
	2		2	2	2	2	

4. Record border equivalences
   - Is the pixel labeled?
     - Yes: Is a border pixel marked?
       - No: No need to update lut
       - Yes: update lut and **update lut initial assigned label too**

Equival

0	1	2	3	4				
0	1	2	3	2	4	6	0	

Why?

			3	
	4	3		
2				

0	1	2	3	4
0	1	2	3	2

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	

		2					
	2					3	
					3	3	
	4		5			3	
	4	4	4			3	
	4						
	4				6	6	
	2		2	2	2	2	

4. Record border equivalences
   - Is the pixel labeled?
     - Yes: Is a border pixel marked?
       - No: No need to update lut
       - Yes: update lut and **update lut initial assigned label too**

Equival								
0	1	2	3	4				
0	1	2	3	2	4	6	0	

Label 3 remains unconnected to label 2

			3	
	4	3		
2				

0	1	2	3	4
0	1	2	3	2

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	

		2					
	2					3	
					3	3	
	4		5			3	
	4	4	4			3	
	4						
	4				6	6	
	2		2	2	2	2	

4. Record border equivalences
   - Is the pixel labeled?
     - Yes: Is a border pixel marked?
       - No: No need to update lut
       - Yes: **First update lut initial label** and then update lut current label

Equival

0	1	2	3	4				
0	1	2	3	2	4	6	0	

Solution: First update lut initial label and then update lut current label

			3	
	4	3		
2				

0	1	2	3	4
0	1	2	2	3

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	

		2					
	2					3	
					3	3	
	4		5			3	
	4	4	4			3	
	4						
	4				6	6	
	2		2	2	2	2	

4. Record border equivalences
   - Is the pixel labeled?
     - Yes: Is a border pixel marked?
       - No: No need to update lut
       - Yes: First update lut initial label and **then update lut current label**

Solution: First update lut initial label and then update lut current label

Equival

0	1	2	3					
0	1	2	3	2	4	6	0	

			3	
	4	3		
2				

0	1	2	3	4
0	1	2	2	2

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	

		2					
	2					3	
					3	3	
	4		5			3	
	4	4	4			3	
	4						
	4				$\boxed{6}$	6	
	2		2	2	2	2	

4. Record border equivalences
   - Is the pixel labeled?
     - Yes: Is a border pixel marked?
       - No: No need to update lut
       - Yes: **First update lut initial label** and **then update lut current label**

Equivalence LUT								
0	1	2	3	4	5	6	7	...
0	1	2	3	2	4	2	0	

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	

		2					
	2					3	
					3	3	
	4		5			3	
	4	4	4			3	
	4						
	4				6	$\boxed{6}$	
	2		2	2	2	2	

4. Record border equivalences
   - Is the pixel labeled?
     - Yes: Is a border pixel marked?
       - No: No need to update lut
       - Yes: **First update lut initial label** and **then update lut current label**

Equivalence LUT								
0	1	2	3	4	5	6	7	...
0	1	2	3	2	4	2	0	

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	

		2					
	2					3	
					3	3	
	4		5			3	
	4	4	4			3	
	4						
	4				6	6	
	2		2	2	2	2	

4. Record border equivalences

Equivalence LUT								
0	1	2	3	4	5	6	7	…
0	1	2	3	2	4	2	0	

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	

		2					
	2					3	
					3	3	
	4		5			3	
	4	4	4			3	
	4						
	4				6	6	
	2		2	2	2	2	

5. Resolve equivalences

Equivalence LUT								
0	1	2	3	4	5	6	7	...
0	1	2	3	2	4	2	0	

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	

		2					
	2					3	
					3	3	
	4		5			3	
	4	4	4			3	
	4						
	4				6	6	
	2		2	2	2	2	

5. Resolve equivalences
   - Assign the border label value (2) to all equivalent labels
   - Skip 0, 1 and 2, because these labels are reserved

Equivalence LUT								
0	1	2	3	4	5	6	7	…
0	1	2	3	2	4	2	0	

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	

		2					
	2					3	
					3	3	
	4		5			3	
	4	4	4			3	
	4						
	4				6	6	
	2		2	2	2	2	

5. Resolve equivalences
   - Assign the border label value (2) to all equivalent labels
   - Skip 0, 1 and 2, because these labels are reserved
   - Remember that the <u>lowest</u> neighbour value was assigned, so resolving from low-to-high will resolve all equivalences

Equivalence LUT								
0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**…**
0	**1**	**2**	**3**	**2**	**4**	**2**	**0**	

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	

		2					
	2					3	
					3	3	
	4		5			3	
	4	4	4			3	
	4						
	4				6	6	
	2		2	2	2	2	

5. Resolve equivalences
   - While not at the end of the lut
     - **Assign the equivalent label**

Equivalence LUT								
0	1	2	3	4	5	6	7	...
0	1	2	3	2	4	2	0	

Label 3 is assigned the label value that is recorded at 3

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	

		2					
	2					3	
					3	3	
	4		5			3	
	4	4	4			3	
	4						
	4				6	6	
	2		2	2	2	2	

5. Resolve equivalences
   - While not at the end of the lut
     - **Assign the equivalent label**

Label 3 is assigned the label value that is recorded at 3

Equivalence LUT								
0	1	2	3	4	5	6	7	...
0	1	2	3	2	4	2	0	

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	

		2					
	2					3	
					3	3	
	4		5			3	
	4	4	4			3	
	4						
	4				6	6	
	2		2	2	2	2	

5. Resolve equivalences
   - While not at the end of the lut
     - **Assign the equivalent label**

Label 4 is assigned the label value that is recorded at 2

Equivalence LUT								
0	1	2	3	4	5	6	7	...
0	1	2	3	2	4	2	0	

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	

		2					
	2					3	
					3	3	
	4		5			3	
	4	4	4			3	
	4						
	4				6	6	
	2		2	2	2	2	

5. Resolve equivalences
   - While not at the end of the lut
     - **Assign the equivalent label**

Label 4 is assigned the label value that is recorded at 2

Equivalence LUT								
0	1	2	3	4	5	6	7	...
0	1	2	3	2	4	2	0	

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	

		2					
	2					3	
					3	3	
	4		5			3	
	4	4	4			3	
	4						
	4				6	6	
	2		2	2	2	2	

5. Resolve equivalences
   - While not at the end of the lut
     - **Assign the equivalent label**

Label 5 is assigned the label value that is recorded at 4

Equivalence LUT								
0	1	2	3	4	5	6	7	…
0	1	2	3	2	4	2	0	

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	

		2					
	2					3	
					3	3	
	4		5			3	
	4	4	4			3	
	4						
	4				6	6	
	2		2	2	2	2	

5. Resolve equivalences
   - While not at the end of the lut
     - **Assign the equivalent label**

Equivalence LUT								
0	1	2	3	4	5	6	7	…
0	1	2	3	2	2	2	0	

Label 5 is assigned the label value that is recorded at 4

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	

		2					
	2					3	
					3	3	
	4		5			3	
	4	4	4			3	
	4						
	4				6	6	
	2		2	2	2	2	

5. Resolve equivalences
   - While not at the end of the lut
     - **Assign the equivalent label**

Label 6 is assigned the label value that is recorded at 2

Equivalence LUT								
0	1	2	3	4	5	6	7	…
0	1	2	3	2	2	2	0	

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	

		2					
	2					3	
					3	3	
	4		5			3	
	4	4	4			3	
	4						
	4				6	6	
	2		2	2	2	2	

5. Resolve equivalences
   - While not at the end of the lut
     - **Assign the equivalent label**

Label 6 is assigned the label value that is recorded at 2

Equivalence LUT								
0	1	2	3	4	5	6	7	…
0	1	2	3	2	2	2	0	

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	

		2					
	2					3	
					3	3	
	4		5			3	
	4	4	4			3	
	4						
	4				6	6	
	2		2	2	2	2	

5. Resolve equivalences
   - While not at the end of the lut
     - Assign the equivalent label

Equivalence LUT								
0	1	2	3	4	5	6	7	...
0	1	2	3	2	2	2	0	

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	

		2					
	2					3	
					3	3	
	4		5			3	
	4	4	4			3	
	4						
	4				6	6	
	2		2	2	2	2	

6. Pass 2: Assign result by using lut

Equivalence LUT								
0	1	2	3	4	5	6	7	...
0	1	2	3	2	2	2	0	

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	

		2					
	2					3	
					3	3	
	4		5			3	
	4	4	4			3	
	4						
	4				6	6	
	2		2	2	2	2	

6. Pass 2: Assign result by using lut
   - Loop all pixels in dst

Equivalence LUT								
0	1	2	3	4	5	6	7	...
0	1	2	3	2	2	2	0	

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	

		2					
	2					3	
					3	3	
	4		5			3	
	4	4	4			3	
	4						
	4				6	6	
	2		2	2	2	2	

6. Pass 2: Assign result by using lut
   - Loop all pixels in dst
     - Pixel is labelled?
     - Yes: Label is equivalent to 2 according to lut?
       - Yes: **Set to 0**

Equivalence LUT								
0	1	2	3	4	5	6	7	...
0	1	2	3	2	2	2	0	

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	

						3	
					3	3	
	4		5			3	
	4	4	4			3	
	4						
	4				6	6	
	2		2	2	2	2	

6. Pass 2: Assign result by using lut
   - Loop all pixels in dst
     - Pixel is labelled?
     - Yes: Label is equivalent to 2 according to lut?
       - Yes: Set to 0
       - No: **Set to 1**

Equivalence LUT								
0	1	2	3	4	5	6	7	...
0	1	2	3	2	2	2	0	

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	

						1	
					1	1	
	4		5			3	
	4	4	4			3	
	4						
	4				6	6	
	2		2	2	2	2	

6. Pass 2: Assign result by using lut
   - Loop all pixels in dst
     - Pixel is labelled?
     - Yes: Label is equivalent to 2 according to lut?
       - Yes: **Set to 0**
       - No: Set to 1

Equivalence LUT								
0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**…**
0	**1**	**2**	**3**	**2**	**2**	**2**	**0**	

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	

						1	
					1	1	
			5			3	
	4	4	4			3	
	4						
	4				6	6	
	2		2	2	2	2	

6. Pass 2: Assign result by using lut
   - Loop all pixels in dst
     - Pixel is labelled?
     - Yes: Label is equivalent to 2 according to lut?
       - Yes: **Set to 0**
       - No: Set to 1

Equivalence LUT								
0	1	2	3	4	5	6	7	…
0	1	2	3	2	2	2	0	

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	

						1	
					1	1	
						3	
	4	4	4			3	
	4						
	4				6	6	
	2		2	2	2	2	

6. Pass 2: Assign result by using lut
   - Loop all pixels in dst
     - Pixel is labelled?
     - Yes: Label is equivalent to 2 according to lut?
       - Yes: Set to 0
       - No: **Set to 1**

Equivalence LUT								
0	1	2	3	4	5	6	7	…
0	1	2	3	2	2	2	0	

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	

						1	
					1	1	
						1	
						1	

6. Pass 2: Assign result by using lut
   - Loop all pixels in dst
     - Pixel is labelled?
     - Yes: Label is equivalent to 2 according to lut?
       - Yes: Set to 0
       - No: Set to 1

Equivalence LUT								
0	1	2	3	4	5	6	7	…
0	1	2	3	2	2	2	0	

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	

						1	
					1	1	
						1	
						1	

7. Cleanup lut
   - By freeing allocated memory

Remove border BLOBs – Two-pass algorithm

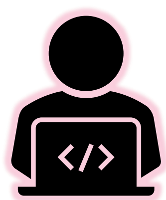
uint32_t **removeBorderBlobsTwoPass**(
const image_t ***src**, image_t ***dst,**
const eConnected **connected**, const uint32_t **lutSize**);

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

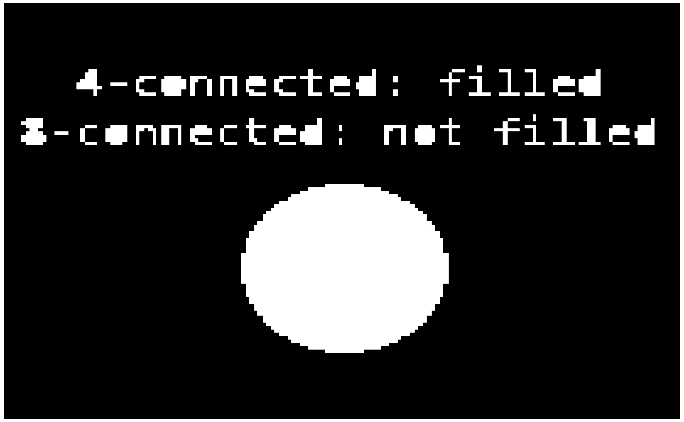
Week 5

1 Morphological filters – removeBorderBlobsTwoPass()

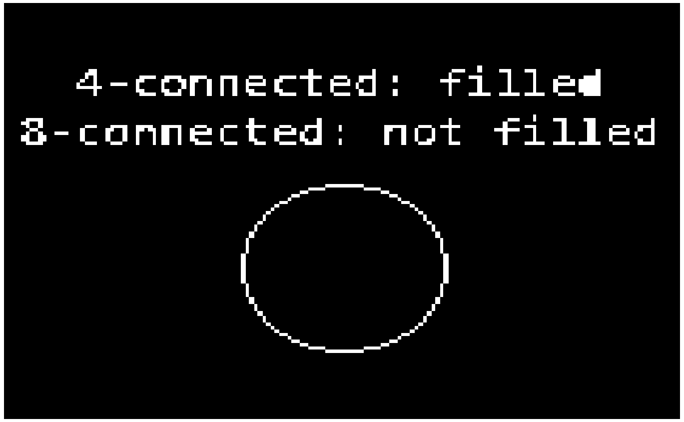
Fill holes

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

Fill holes - example



4-connected



8-connected

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:
  - a hole is defined as not being connected to the background

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:
  - a hole is defined 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);
```

Fill holes – Two-pass algorithm

Tips

- The same 7 steps are applicable

1. Initialize lookup table (lut)
2. Mark the border pixels in the destination
3. Pass 1: Label the objects, skipping the borders, and record equivalences in lut
4. Record border equivalences
5. Resolve equivalences
6. Pass 2: Assign result by using lut
7. Cleanup lut

Fill holes – Two-pass algorithm

Tips

- No changes

1. Initialize lookup table (lut)
2. Mark the border pixels in the destination
3. Pass 1: Label the objects, skipping the borders, and record equivalences in lut
4. Record border equivalences
5. Resolve equivalences
6. Pass 2: Assign result by using lut
7. Cleanup lut

Fill holes – Two-pass algorithm

Tips

- Instead of marking the object pixels, mark the background pixels

1. Initialize lookup table (lut)
2. Mark the border pixels in the destination
3. Pass 1: Label the objects, skipping the borders, and record equivalences in lut
4. Record border equivalences
5. Resolve equivalences
6. Pass 2: Assign result by using lut
7. Cleanup lut

Fill holes – Two-pass algorithm

Tips

- Instead of labelling the objects, label the background

1. Initialize lookup table (lut)
2. Mark the border pixels in the destination
3. Pass 1: Label the objects, skipping the borders, and record equivalences in lut
4. Record border equivalences
5. Resolve equivalences
6. Pass 2: Assign result by using lut
7. Cleanup lut

Fill holes – Two-pass algorithm

Tips

- No changes

1. Initialize lookup table (lut)
2. Mark the border pixels in the destination
3. Pass 1: Label the objects, skipping the borders, and record equivalences in lut
4. Record border equivalences
5. Resolve equivalences
6. Pass 2: Assign result by using lut
7. Cleanup lut

Fill holes – Two-pass algorithm

Tips

- No changes

1. Initialize lookup table (lut)
2. Mark the border pixels in the destination
3. Pass 1: Label the objects, skipping the borders, and record equivalences in lut
4. Record border equivalences
5. Resolve equivalences
6. Pass 2: Assign result by using lut
7. Cleanup lut

Fill holes – Two-pass algorithm

Tips

- Loop all pixels in dst
- Pixel is background?
  - Yes: Set to 1 (fills the holes)
  - No: Label is equivalent to 2 according to lut?
    - Yes: Set to 0 (because it is connected to a border)
    - No: Set to 1 (because it is an object)

1. Initialize lookup table (lut)
2. Mark the border pixels in the destination
3. Pass 1: Label the objects, skipping the borders, and record equivalences in lut
4. Record border equivalences
5. Resolve equivalences
6. Pass 2: Assign result by using lut
7. Cleanup lut

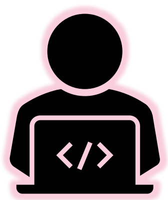
Fill holes – Two-pass algorithm

Tips

- No changes

1. Initialize lookup table (lut)
2. Mark the border pixels in the destination
3. Pass 1: Label the objects, skipping the borders, and record equivalences in lut
4. Record border equivalences
5. Resolve equivalences
6. Pass 2: Assign result by using lut
7. Cleanup lut

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

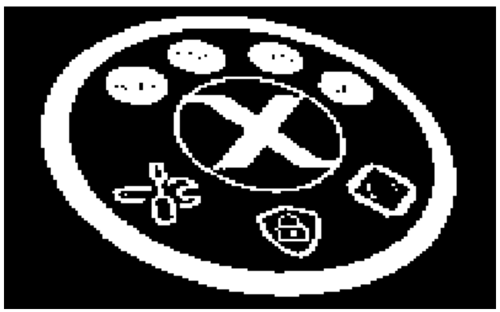
$$close(A, B) = (A \oplus B) \ominus B$$

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

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

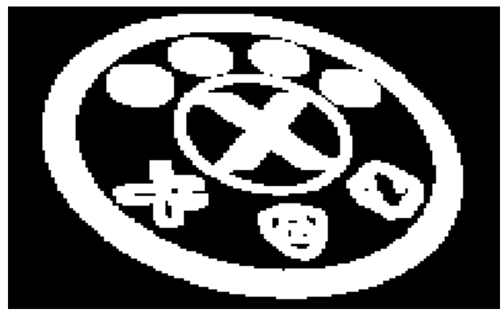
Examples

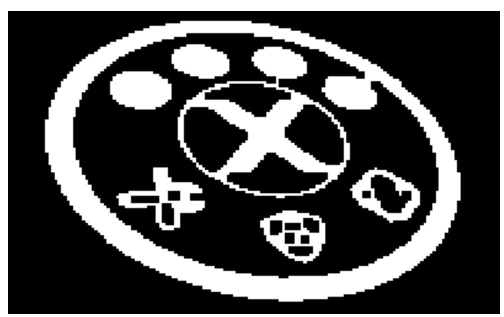
A

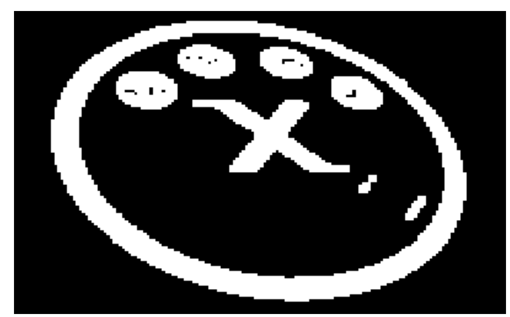


B

1	1	1
1	1	1
1	1	1

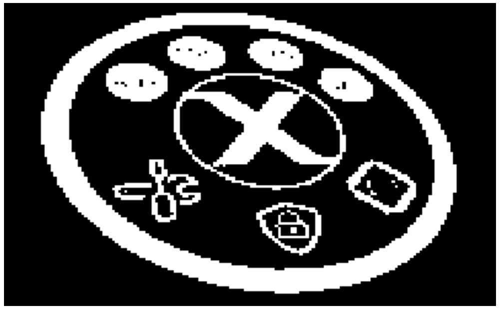






Examples

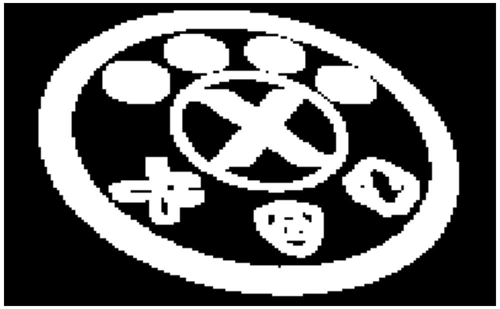
A



B

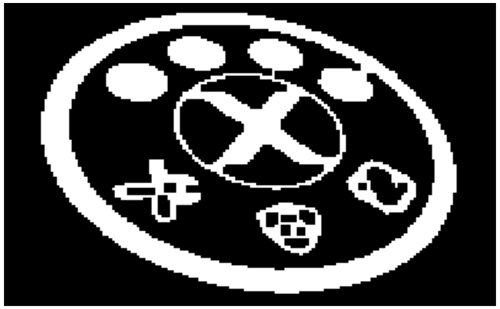
1	1	1
1	1	1
1	1	1

Dilation

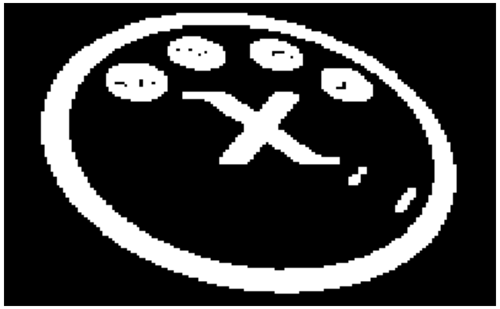


Erosion

Closing

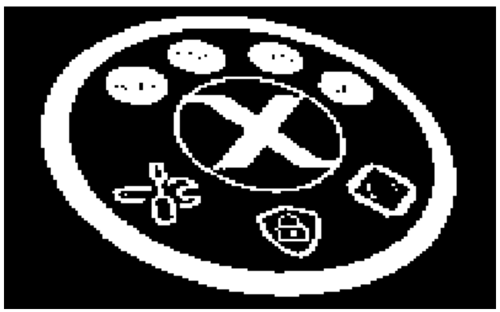


Opening



Examples

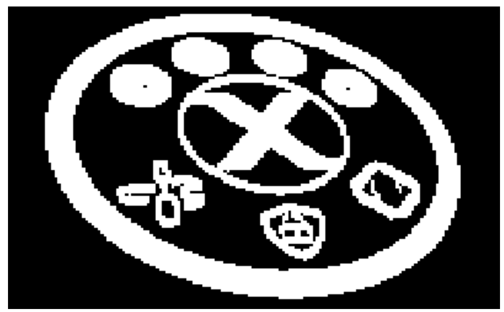
A



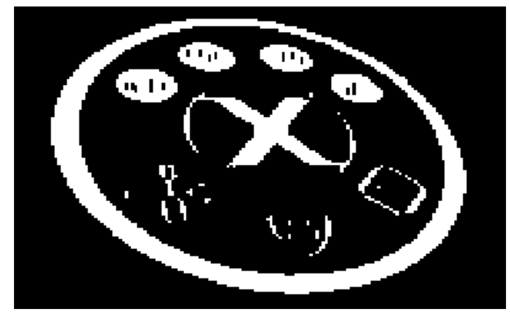
B

0	1	0
0	1	0
0	1	0

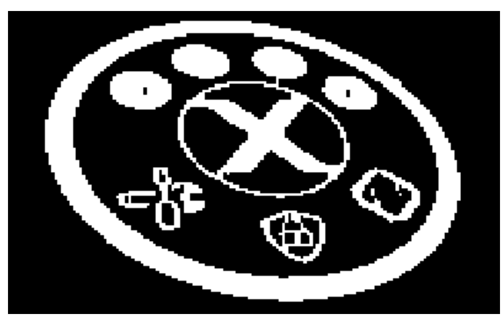
Dilation



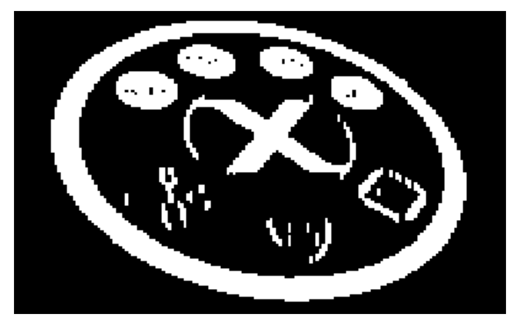
Erosion



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:

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

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

$open(erode_0(A,B),B)$

$A = erode_0(A,B)$

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

$$\overbrace{\qquad\qquad\qquad\qquad}^{open(erode_1(A,B),B)}$$

$erode_1(A,B)$

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

$$\overbrace{\hspace{6cm}}^{open(erode_2(A,B),B)}$$

$erode_2(A,B)$

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

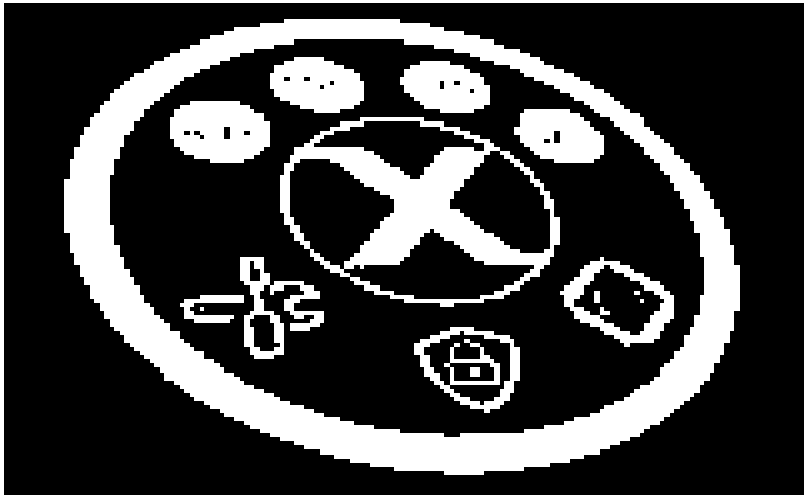
$$K_1(A) \cup K_2(A) \cup K_3(A)$$

Binary skeleton - example

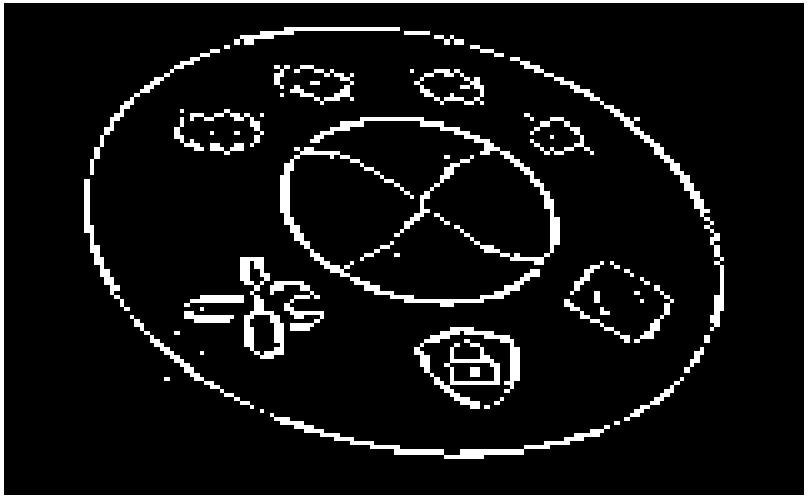
B

0	1	0
1	1	1
0	1	0

A



$K_n(A)$

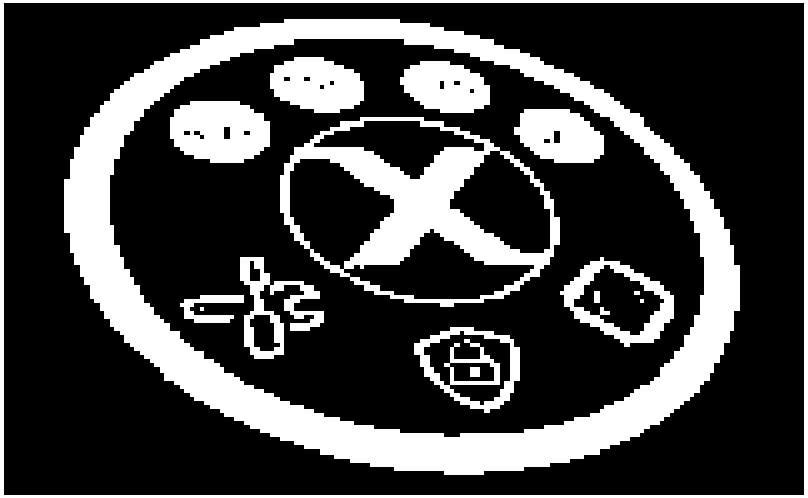


Binary skeleton - example

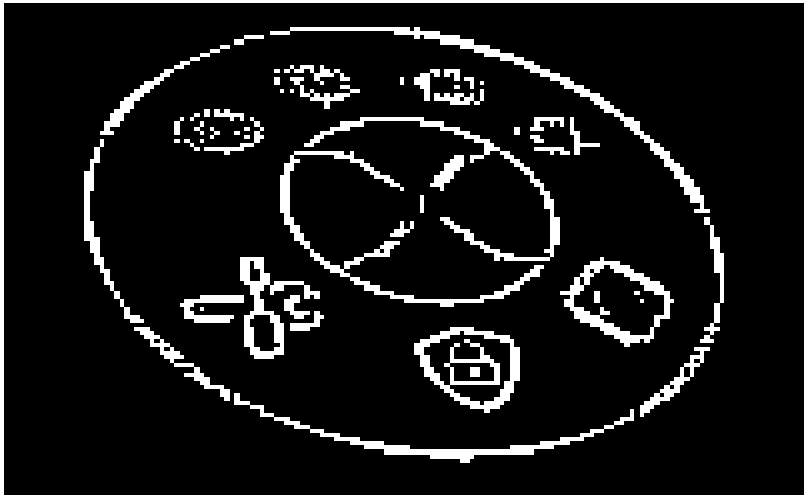
B

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

0	0	0
1	1	0
0	0	0

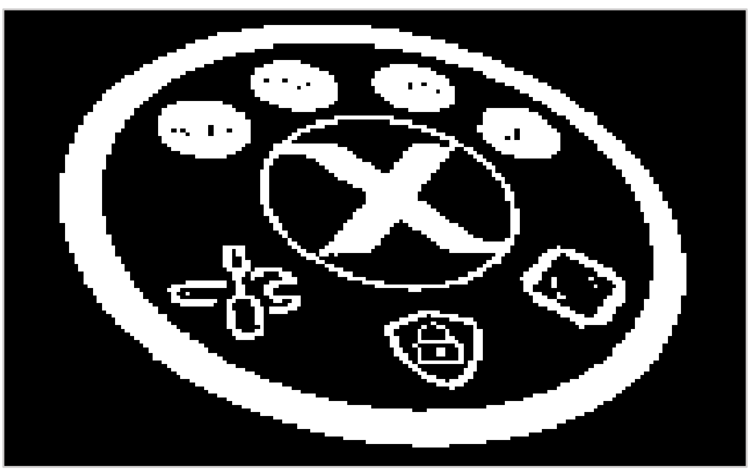
C

0	0	0
0	0	1
0	0	0

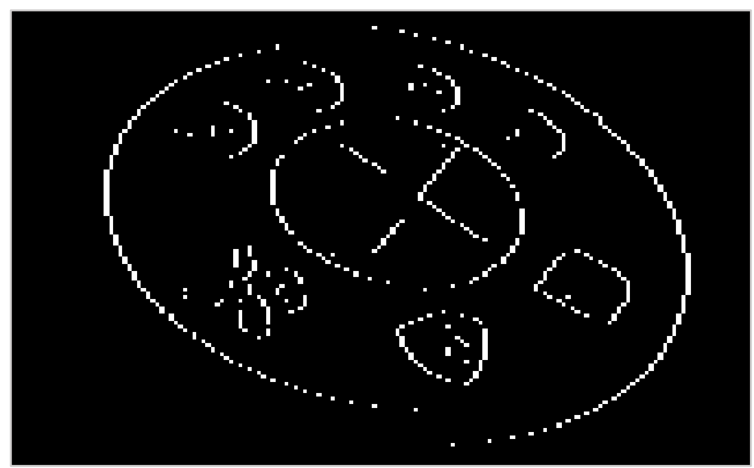
Alternative representation

-	-	-
1	1	0
-	-	-

A



$hitmiss(A, B, C)$



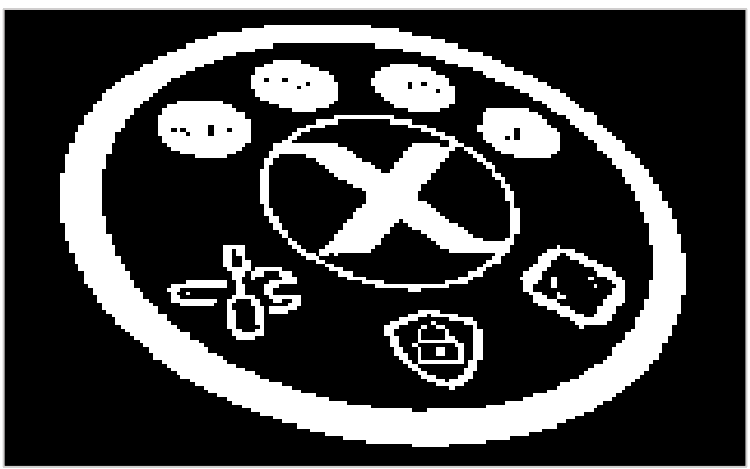
Binary hit-miss - example

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

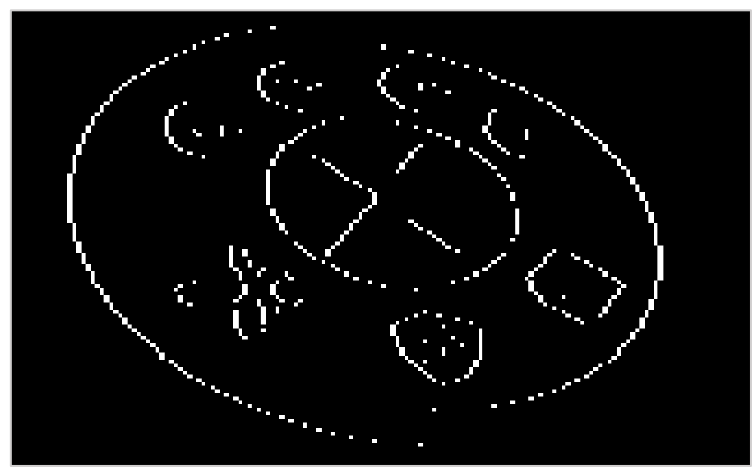
Alternative representation

-	-	-
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

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

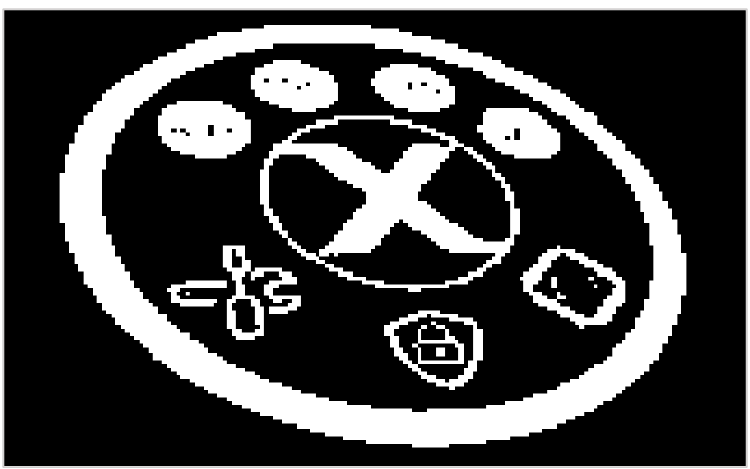
where

B is the structuring mask

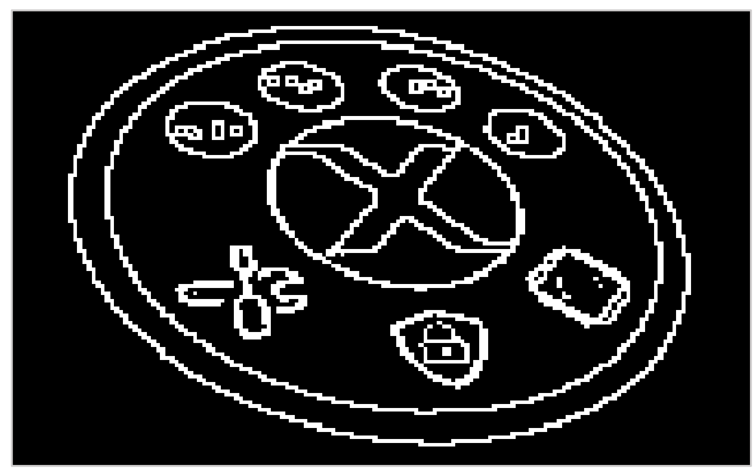
Binary outline - example

$$B = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

A



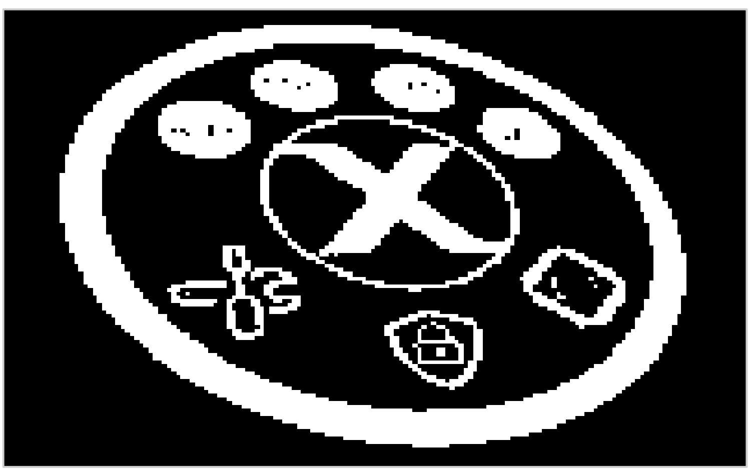
$outline(A, B) = A - (A \ominus B)$



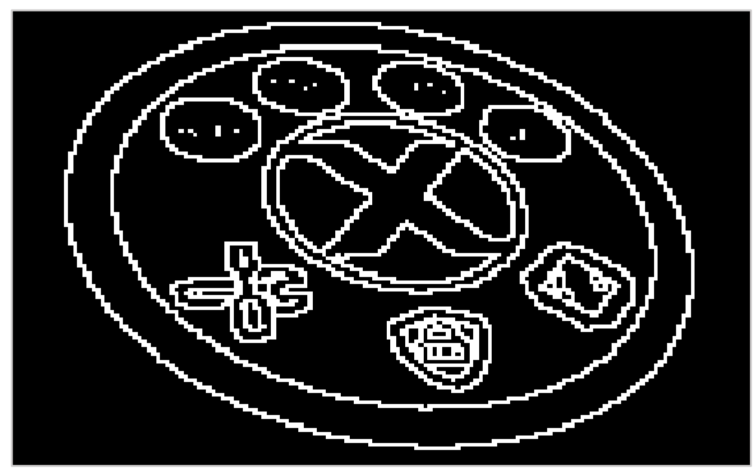
Binary outline - example

$$B = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

A



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

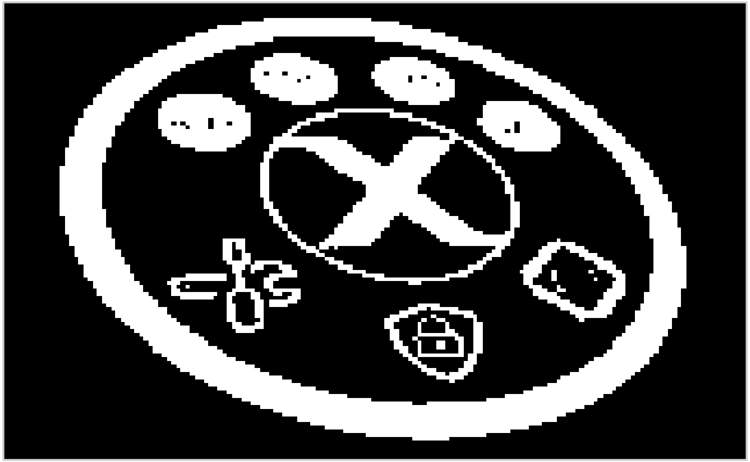


Binary outline - example

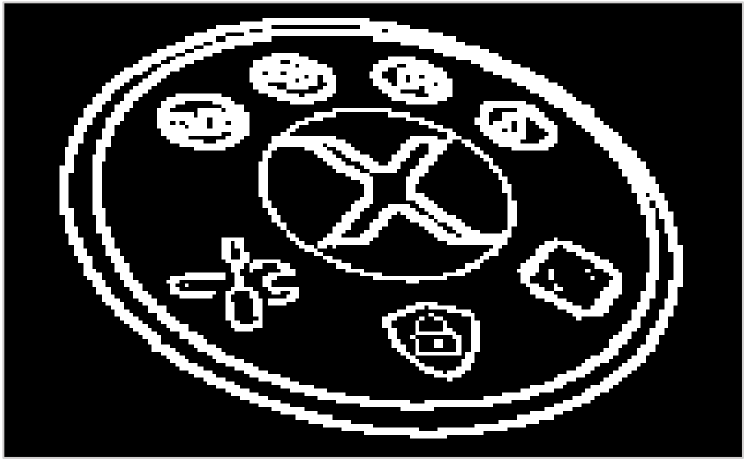
B

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



$outline(A, B)$

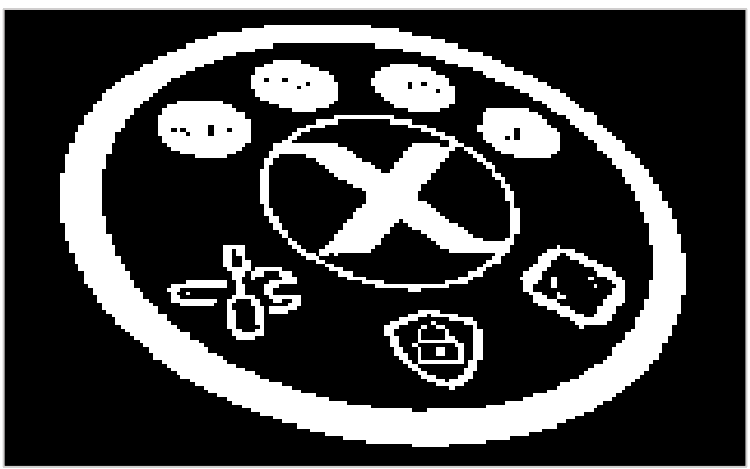


HAN_UNIVERSITY OF APPLIED SCIENCES

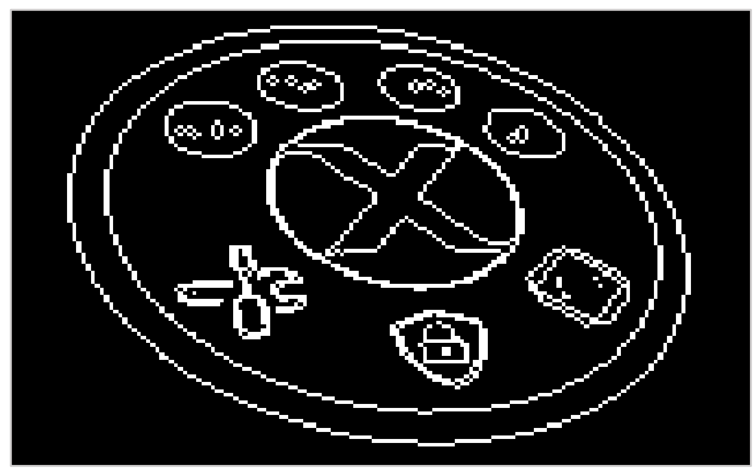
Binary outline - example

$$B = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

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**

References

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