C:\opencv\build\x64\vc16\lib
opencv_world4110d.lib
C:\opencv\build\include
D:/FresherXavisTech/Image/

https://www.di.univr.it/documenti/OccorrenzaIns/matdid/matdid699113.pdf

Slide 1:

The Hit-or-Miss Transform (HMT) is a method in binary image processing used to **find specific shapes or patterns** in an image.

The operation uses **two simultaneous erosions**:

- One erosion is applied to the original image using a structuring element to ensure the pattern **fits perfectly** in the foreground.
- At the same time, we erode the inverse (complement) of the image using a different structuring element, to make sure the background area matches the parts that should not be part of the shape — the 'Miss' areas.

The **final result** is the **intersection of the two erosions**. If both the foreground and background match at a certain position, it is identified as a perfect match for the target pattern.

Example illustrating the concepts of HMT:

In this example, the input binary image A contains **three separate foreground regions**: C, D, and E.

- Here, **D is the small square object we want to detect**.
- C and E are unwanted structures or noise.
- Each region includes a **black dot** indicating the **origin** of the object.

To locate D in the foreground, we perform **erosion of A using a structuring element B1**, where **B1 is equal in shape to D**.

The figure shows that this erosion returns a **single point** at the origin of D – which is what we expect. **However, it also includes parts of region C**, because **C is larger than D**, so the shape of D **can fit inside C** in some places.

This demonstrates that just checking the foreground is not enough - we might make mistakes and detect the wrong object.

To find the **true location of D**, we must also consider the **background**.

We take the **complement of image A**, meaning all foreground pixels become background and vice versa.

Then, we **erode the complemented image with a structuring element B2**, designed to fit the **border of D in the background**.

B2 consists of a **rectangle one pixel thick**, surrounding the size of D.

As shown in the figure, the erosion result includes the **origin of D**, but may also include parts of C and the part of complement of A.

In the end, **only one point is common** between the two erosion results — this point is exactly the **origin of object D**.

Therefore, the **intersection of these two erosions** is the final output of the **Hit-or-Miss Transform.** It tells us **exactly where the shape D is located** in the image.

Slide 2:

This is an example I programmed, handling the parameters of the structuring element similarly to the OpenCV library, where 1 represents the background, -1 represents the foreground, and 0 means 'don't care

We can see that this structure matches at these positions here and here, so I obtained the results here and here

Similarly, with the two structure elements below.

Slide 3:

This is also a similar example.

Slide 4:

Boundary extraction is a key technique in image processing used for:

- Separating objects from the background.
- Detecting object shapes and contours.
- Preprocessing for recognition or object detection tasks.

There are two common methods:

The first method is to subtract the eroded image from the original image. During erosion, the object shrinks as pixels near the edges are removed. The lost pixels represent the object's boundary. Therefore, by subtracting the eroded image from the original, we obtain a thin outline around the object, called the outer boundary. This boundary helps clearly define the shape and edges of the object.

The second method is the morphological gradient. This technique extracts the boundary by subtracting the eroded image from the dilated image. During dilation, the object expands outward, while during erosion, it shrinks inward. The difference between these two results represents the boundary region, including both the outer and inner edges. Thus, the gradient gives a clearer and thicker outline that highlights the entire boundary of the object — both inside and outside the original shape.

Slide 25:

These are the results I obtained when applying the two boundary extraction methods. We can see that the method using only erosion produces a thin boundary line, and the boundary lies entirely inside the original object.

When using the second method, the boundary is twice as thick as that obtained by erosion alone, due to the additional use of dilation. The boundary extends both inside and outside the original object.

Slide 26:

Thinning is a morphological operation that preserves the structure and connectivity of an object while removing edge pixels. The result is a skeletal representation, or the 'skeleton', of the original shape.

The thinning operation uses the hit-or-miss transform with 8 different structuring element patterns. Each structuring element represents a specific direction (0° degree, 45°, 90°, ..., 315°).

When performing the thinning operation with a structuring element, a thin layer at the edge of the object is removed in the direction corresponding to the structuring element. This is the result of thinning using the first SE, which corresponds to 0 degrees. We can see that a thin layer at the top has been removed.

After performing the thinning operation using 8 structuring elements corresponding to different directions, the outer layers of the original object are iteratively removed along all 8 directions without breaking its connectivity or overall structure. This process is repeated until no further changes occur, resulting in the final thinned object.

Slide 27:

This is the flowchart of my program to implement the thinning operation. First, the thinning operation formula is applied using 8 structuring elements. Then, the result is checked to see if there are any changes. If there are changes, the process repeats; otherwise, it returns the final result.

on the right is the result of the program. The original object is gradually eroded step by step, and the final result is obtained when no further changes occur.

Slide 28:

Skeletonization is a morphological process that reduces a binary object to its central structure, known as the **skeleton**.

The skeleton preserves both the shape and connectivity of the original object, while minimizing its thickness—usually to a one-pixel-wide representation.
In this presentation, I will explain how to implement skeletonization using two fundamental morphological operations: **erosion** and **opening**.

The morphological skeleton S(A) of a binary image A can be defined using the following formula:

At each iteration k, we erode the image and remove the parts that are smoothed out by opening. What remains is the thin central structure—the skeleton layer at that level. Repeat the process until the image is completely eroded—that is, all foreground pixels are removed.

The union of all skeleton fragments gives us the final skeletonized image.

Here we have a simple illustration of the skeletonization process.

Slide 29:

Here’s a step-by-step explanation of the algorithm:

1. Start with the original binary image.
2. Erode the image using the structuring element.
3. Perform opening on the eroded image.
4. Subtract the opened image from the eroded image. This gives us the skeleton fragment for this iteration.
5. Accumulate the result using a bitwise OR with the current skeleton.
6. Repeat the process until the image is completely eroded—that is, all foreground pixels are removed.

This is the original image we are analyzing. The first column shows the results of successive erosion iterations. Note that after the second iteration, the result becomes an empty set, so we only consider two iterations. The second column shows the opening by the structuring element **B** of the eroded images in the first column. The third column displays the result of the difference between the first and second columns—these are the skeleton fragments at each iteration. The fourth column represents the union of all skeleton fragments across iterations. The final image here is the result of the skeletonization operation. This result is **not connected** and contains regions thicker than one pixel, as we observed in the previous example.

Slide 30:

This is the result of applying thinning and skeletonization to the fingerprint image processing task I worked on in our previous session.
We can see that the thinning result has made the ridge lines thinner, but there are still many unwanted or extra lines.

The skeletonization result appears to be better — it doesn’t have those extra lines and represents the fingerprint shape more accurately.

However, the ridge lines are not continuous, as we discussed in the previous slide.

Slide 31:

Here are some OpenCV functions used to perform morphological transformations.

Slide 32:

This function is used to perform morphological transformations. We can apply different types of transformations through the operation parameter

Slide 33:

I encountered a problem when using the histogram comparison function with this image. My expected result is shown on the right, but when I applied OpenCV's function, the result was not as expected.

I think the issue is caused by the image padding process in the function. When I set the parameters this way, it uses BORDER_CONSTANT with the default border value. I tried setting the border value to 255 and 0, but the result still did not match my expectation. Interestingly, when the border value is set to 255, the result is the same as using the default value. To clarify this, I will dive deeper into the hit-or-miss process.

Slide:

According to the theory, we pad the original image with 0 and the complement image with 255, then apply erosion to both images and compute their intersection to obtain the result. With this approach, I was able to get the expected outcome.

Slide:

The reason for the incorrect result when I used OpenCV's function is that setting border = 0 causes both the original image and its complement to be padded with the same value — 0. In other words, the complement image is padded with the same value as the original. I have simulated the process that leads to this outcome here. The same issue occurs when I set the border to 255. The solution to this problem is to manually pad the images: pad the original image with 0 and the complement image with 255, then perform erosion on each separately and finally compute their intersection\

I built a program based on the theory and wrapped it in a class, as shown below.