Binary Images

Shree K. Nayar

Monograph: FPCV-1-3

Module: Imaging

Series: First Principles of Computer Vision

Computer Science, Columbia University

March 01, 2022

FPCV Channel

FPCV Website

Binary Images

Shree K. Nayar
Columbia University

Topic: Binary Images, Module: Imaging
First Principles of Computer Vision

1

Binary Images

Binary Image: Can have only two values (0 or 1). Simple to process and analyze.

2

Binary images are the simplest type of images used in computer vision. A binary image can have one of two values: 0 or 1. For instance, in the image shown on the right, the value of 1 (white) represents the object while the value of 0 (black) denotes the background. These images are very easy to process, store, and analyze. Despite their simplicity, they are very useful — there are many vision tasks, especially in structured environments such as assembly lines in factories, that can be performed efficiently and robustly using binary images.

In order to create a binary image, we start with a gray-level image, with values that range from 0 to *N*. We then threshold it with a value *T* that is either automatically computed from the image or chosen by the user. The result is a characteristic or indicator function, *b(x,y)*, for which the output value is 0 if the corresponding value in the gray-level image is less than the threshold and 1 otherwise (it can also be the other way around).

Making Binary Images

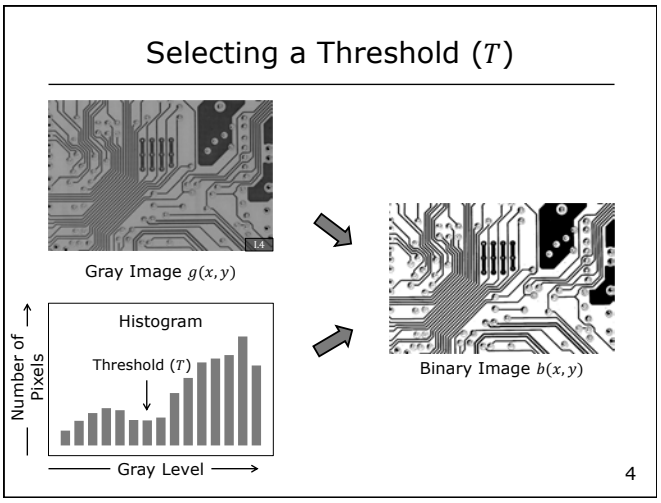
Binary Image $b(x,y)$: Usually obtained from Gray-level image $g(x,y)$ by Thresholding.

Characteristic Function:

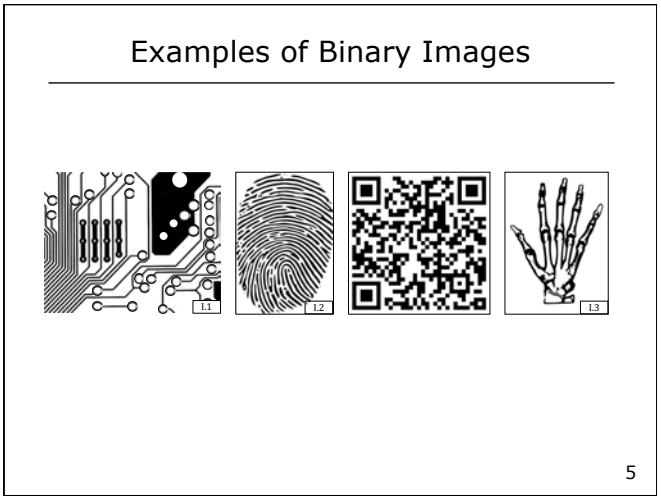
$$b(x,y) = \begin{cases} 0, & g(x,y) < T \\ 1, & g(x,y) \geq T \end{cases}$$

3

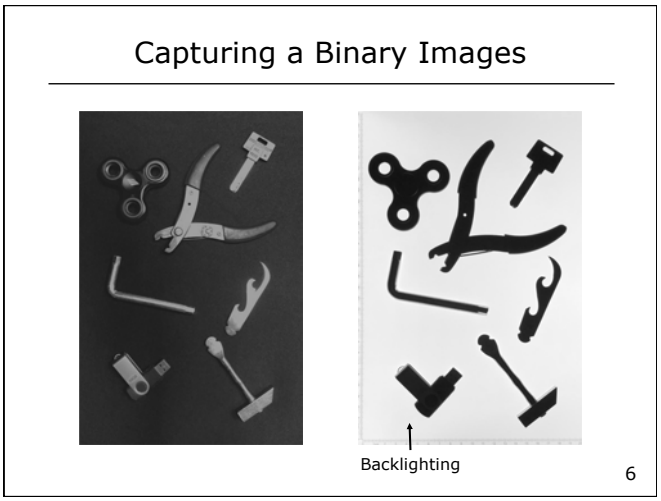
In order to select the threshold *T*, we can compute a histogram of the original gray-level image. In this example, the gray-level image has an 8-bit brightness value at each pixel. Hence, the horizontal axis of the histogram has gray levels that go from 0 to 255. For each gray level, the histogram shows the number of pixels in the image that have that level. The histogram seen here has two modes (or peaks), one corresponding to the background and the other to the objects in the foreground. The best choice for the threshold *T* would be a value that lies in the valley between the two modes.



Here are a few of the many applications of binary images. They are widely used for detecting defects in printed circuit boards, such as breaks in copper strips. In the case of fingerprint analysis, a special type of lighting is used to enhance the curves and ridges in the image of the finger, which is then converted to a binary image. Another popular application is the detection and decoding of visual codes like QR codes. Binary images are also used in other domains such as medical image analysis.



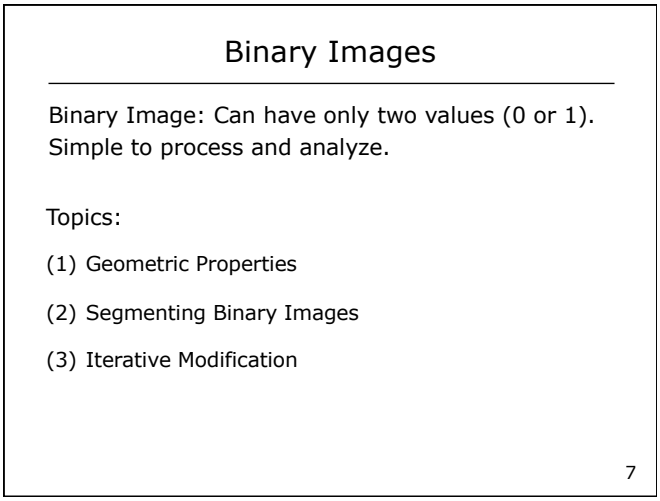
Binary images are also useful in imaging three-dimensional (3D) objects. Irrespective of how complex a 3D object is, if we drop it on a horizontal plane, it will land in one of a finite number of stable configurations. Thus, if we have 3D objects sitting on a plane, as shown here, and the plane is imaged from the top, then the camera will see each of the objects in one of its stable configurations. In any given stable configuration, the 3D object produces a 2D shape that may be translated or rotated in the image. If we can develop an algorithm to recognize



a 2D shape even when it is translated or rotated in the image, we can recognize the corresponding 3D object.

In the example shown here, however, it is difficult to threshold the image on the left because the objects are made of different materials. This results in object regions that can be bright or dark and can include shading, texture and highlights. In applications where lighting can be controlled, backlighting can be utilized where the objects sit on a translucent surface that is lit from beneath. When this is done, the image captured (right) by the camera is virtually a binary one, even without any processing.

In this lecture, we will discuss the processing of binary images. First, we will discuss the geometric properties of a binary image of a single object. We will show that some of these properties are invariant to translation and rotation of the object on the plane it sits, hence they can be used to recognize the object.



When the binary image includes multiple objects, the objects need to be assigned different labels before the geometric properties of each one can be computed. This labeling problem is referred to as segmentation. We will present algorithms for efficiently segmenting a binary image into different objects.

Finally, we will talk about iterative modification. This is a class of algorithms that change a pixel value (0 to 1 or 1 to 0) based solely on the values of its immediate neighbors. In doing so, care must be taken to ensure that the structure of the object is preserved and that new bodies or holes are not introduced into the image. Such modification algorithms are inherently parallelizable — the pixel modifications can be applied to large numbers of pixels, simultaneously. These algorithms can also be iterative — the modification can be applied to an image repeatedly. This approach can be used to, for instance, quickly thin an object down to its skeleton.

Geometric Properties

Shree K. Nayar
Columbia University

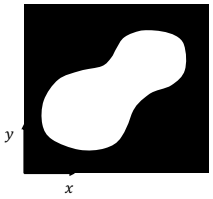
Topic: Binary Images, Module: Imaging
First Principles of Computer Vision

8

Geometric Properties of Binary Images

Assume:

- $b(x,y)$ is continuous
- Only one object



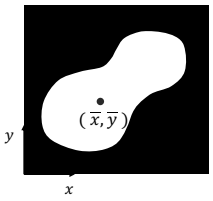
9

Let us examine how the geometric properties of a binary image can be computed. Assume that the binary image shown here is continuous with spatial coordinates *x* and *y*; we will discuss discrete binary images later. Let us also assume, for now, that there is only one object in the binary image. Our characteristic function *b(x,y)* is 1 for points on the object and 0 for points in the background.

Area and Position

Area: (Zeroth Moment)

$$A = \iint_I b(x,y)\,dx\,dy$$



Position: Center of Area (First Moment)

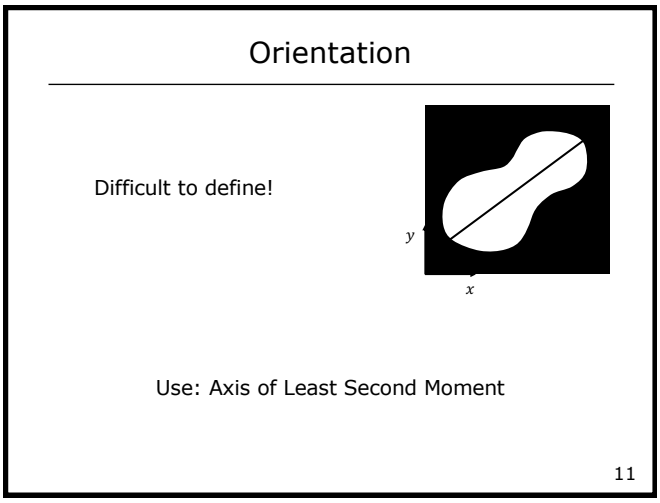
$$\bar{x} = \frac{1}{A}\iint_I x\,b(x,y)\,dx\,dy \quad , \quad \bar{y} = \frac{1}{A}\iint_I y\,b(x,y)\,dx\,dy$$

10

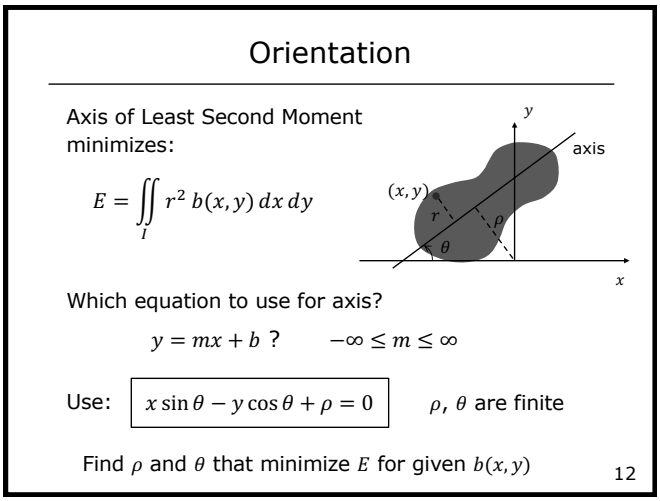
The simplest geometry property we can compute is the area of the object, which is the zeroth moment. The area is computed by integrating the characteristic function *b(x,y)* over the entire image. The area is a useful property because it is sometimes sufficient for distinguishing between a small number of objects.

Another useful property of an object is its location. If, for instance, we have an application in which a robot needs to pick up an object, the robot would need to know the object's location. A straightforward way to determine its location is by finding the first moments of its binary image. These can be found by computing the integral of *x* times *b(x,y)* over the entire image divided by the area to get $\bar{x}$ and the integral of *y* times *b(x,y)* over the entire image divided by the area to get $\bar{y}$. These are the *x* and *y* coordinates of the center of the area. This center is analogous to the "centroid" of an object in mechanics. Given an object with uniform thickness and material composition, the location of its centroid, or center of mass, corresponds to the center of its area.

Returning to the robot example, our robot would also need to know the orientation of the object before it can grasp it. Orientation is a more nuanced concept, as one can think of many ways to define it. For our purposes here, we need to find an axis that is well-defined for any given shape, irrespective of its position and rotation in the image. That axis is the axis of least second moment. Here is one way to gain physical intuition for what this axis means. Imagine the object to be a thin sheet of uniform material and thickness. If we were to spin the object about the axis of least second moment, it would require less effort than spinning it about any other axis.



How do we find the axis of least second moment? The second moment (*E)* for any chosen axis is the sum of the square of the shortest distance (*r)* of each point on the object from that axis. In order to define the chosen axis, we could use the straight-line equation *y = mx + b*. This representation of the line, however, is problematic for our purposes since *m* goes from minus infinity to infinity. Instead, we will use a different parametrization of the straight line: *x sin* θ - *y cos* θ + *ρ = 0*. θ is the angle between the line and the horizontal axis, which can only range from 0 to 2π. *ρ* is the perpendicular distance from the line to the origin. Note that *ρ* also has to be finite because the object (and hence its axis of least second moment) must lie within the image. Our goal is to find the *ρ* and θ that minimize *E* for any given binary image. For now, we are once again assuming that there is only one object in the binary image.



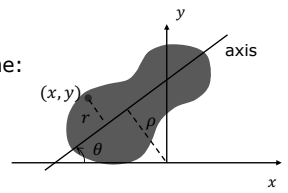
Here is a math primer that will allow us to review some properties of a straight line. If we are given a line *ax + by + c = 0*, the distance *r* of a point *(x,y)* from that line is the absolute value of *ax + by + c* divided by the square root of $a^2 + b^2$. From our line expression *x sin* θ - *y cos* θ + *ρ = 0,* we can determine *a*, *b*, and *c* and plug them into the equation for *r*. In the denominator we have $\sin^2\theta + \cos^2\theta$ which equals 1. Therefore, *r* is equal to this expression 1 which actually turns out to be the left-hand side of our straight-line equation.

Distance Between Point and Line

Given a line $ax + by + c = 0$

Distance of point (x, y) from line:

$$r = \left| \frac{ax + by + c}{\sqrt{a^2 + b^2}} \right|$$



Similarly, given axis $x \sin\theta - y \cos\theta + \rho = 0$

Distance of point (x, y) from axis:

$$r = \left| \frac{x \sin\theta - y \cos\theta + \rho}{\sqrt{\sin^2\theta + \cos^2\theta}} \right|$$

$$\boxed{r = |x \sin\theta - y \cos\theta + \rho|} \quad \boxed{1}$$

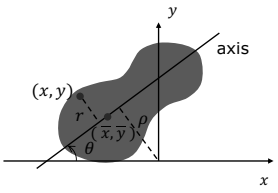
MATH PRIMER 13

We plug this expression for *r* back into the equation for the second moment *E*. In order to minimize *E*, we find the derivative of *E* with respect to *ρ* and set it equal to zero. By doing so, we obtain a very simple expression, which is the area multiplied by $\bar{x}$ sin θ - $\bar{y}$ cos θ + *ρ* is equal to zero, where ($\bar{x}$, $\bar{y}$) is again the center of the object. This tells us the axis that corresponds to the least second moment must pass through the center of the object.

Minimizing Second Moment

Axis of Least Second Moment minimizes:

$$E = \iint_I r^2 \, b(x, y) \, dx \, dy$$



So, minimize:

$$E = \iint_I (x \sin\theta - y \cos\theta + \rho)^2 b(x, y) \, dx \, dy$$

Using $\frac{\partial E}{\partial \rho} = 0$ we get: $A(\underline{\bar{x} \sin\theta - \bar{y} \cos\theta + \rho}) = 0$

Axis passes through center $(\bar{x}, \bar{y})$!

14

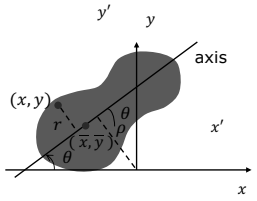
To remove the parameter *ρ* from our expression for *E,* we will shift the coordinate frame of the image so that it lies at the center, ($\bar{x}$, $\bar{y}$), of the object. We will define *x'* as *x* - $\bar{x}$, and *y'* as *y* - $\bar{y}$. Substituting in our previous expression for *E*, we obtain this expression 1 in which *a, b*, and *c* are constants. These constants correspond to double integrals. In the expression for *a*, the $(x')^2$ implies that it is the second moment about the *y*-axis. In the expression for *b*, the (*x'y'*) is a cross term and *b* is referred to as the product moment. The values *a*, *b*, and *c* are easy to compute from the image. By shifting the coordinate frame to the center, we have essentially removed *ρ* from the expression for *E.*

Shift the Coordinate System

Change coordinates:

$x' = x - \bar{x}$, $y' = y - \bar{y}$

$x \sin\theta - y \cos\theta + \rho$
$= x' \sin\theta - y' \cos\theta$



Therefore, we can rewrite E as:

$$\boxed{E = a \sin^2\theta - b \sin\theta \cos\theta + c \cos^2\theta}$$

1

where:

$$a = \iint_{I'} (x')^2 \, b(x, y) \, dx' \, dy'$$

$$b = 2 \iint_{I'} (x'y') \, b(x, y) \, dx' \, dy'$$

$$c = \iint_{I'} (y')^2 \, b(x, y) \, dx' \, dy'$$

(a, b, c are easy to compute)

15

With ρ out of the picture, we can minimize *E* with respect to θ to obtain this simple expression [1], which states that tan 2θ is equal to *b* divided by *a* - *c*. There are actually two solutions to θ that result from this expression, because tan 2θ is equal to tan 2θ + π. If we multiply the numerator and denominator of this expression by -1, we will still have the same value.

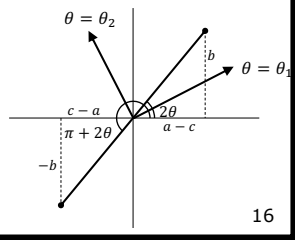
Finally, Minimize E

Using $\frac{dE}{d\theta} = (a-c)\sin 2\theta - b\cos 2\theta = 0$ [1] we get: $\boxed{\tan 2\theta = \frac{b}{a-c}}$

We know that: $\tan 2\theta = \tan(2\theta + \pi) = \frac{-b}{c-a}$

θ has two solutions.

1. $\theta = \theta_1$
2. $\theta = \theta_2 = \theta_1 + \frac{\pi}{2}$



One gives Minimum of E
and the other Maximum of E

16

The illustration on the right provides a geometrical interpretation of the two solutions. Take a look at the triangle on the right with *a* - *c* as the base and *b* as the height. In this case, tan 2θ is equal to *b* divided by *a* - *c*. We can form a second triangle with an angle of 2θ + π, which is a reflection of the first triangle. From this triangle we get tan 2θ + π is equal to -*b* divided by *c* – *a*, which is the same value we got for tan 2θ using the first triangle.

What do the two solutions to θ represent? One solution maximizes *E* while the other minimizes *E*. The two solutions θ_1 and θ_2 are perpendicular to each other. Thus, the axis of minimum inertia will be perpendicular to the axis of maximum inertia. To find the axis of minimum inertia, we can take the second derivative of *E* with respect to θ. Among the two solutions θ_1 and θ_2, the one for which the second derivative of *E* is greater than zero corresponds to the axis of minimum second moment.

Which One To Use?

Using second derivative test:

If $\frac{d^2E}{d\theta^2} = (a-c)\cos 2\theta + b\sin 2\theta$ { > 0 then Minimum; < 0 then Maximum

Substituting $\cos 2\theta_1$, $\sin 2\theta_1$, $\cos 2\theta_2$ and $\sin 2\theta_2$:

$$\frac{d^2E}{d\theta^2}(\theta_1) > 0 \quad \text{and} \quad \frac{d^2E}{d\theta^2}(\theta_2) < 0$$

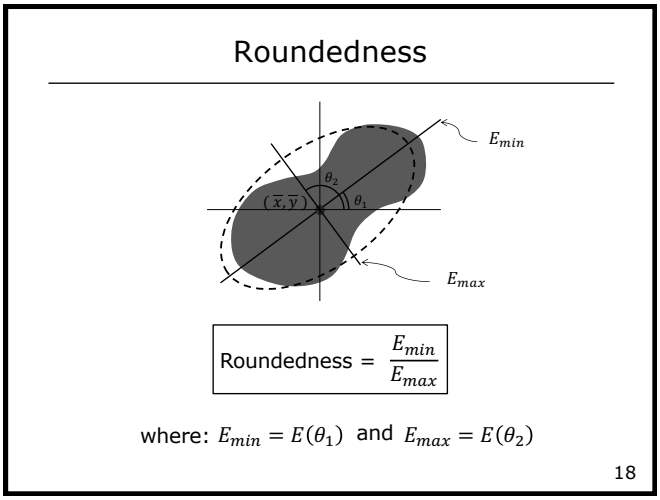
Therefore,

Orientation: $\boxed{\theta = \theta_1 = \frac{atan2(b, a-c)}{2}}$

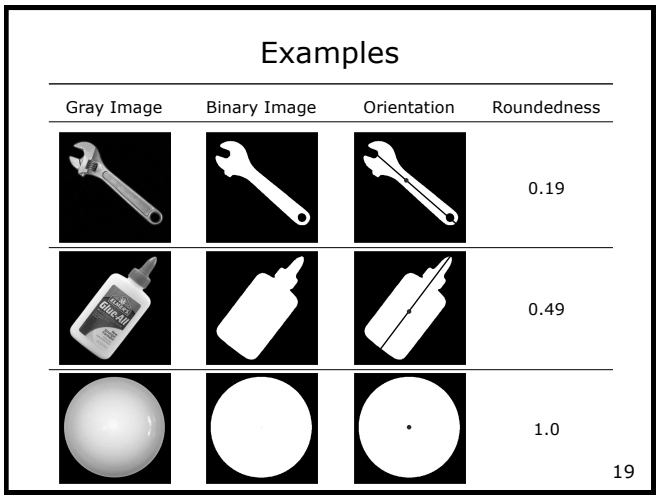
17

If we substitute the two solutions into the second derivative of *E,* we will find that for θ_1, the second derivative is greater than zero; that is the solution we are looking for. That is, θ_1 is the orientation of the axis of least second moment. As for its position, we already know it passes through the center (*x*′*y*′) of the object. Equally important is the fact that the values of *E* corresponding to θ_1 and θ_2 are useful geometric properties of the object that are invariant to its position and orientation.

We can also get a measure of how "rounded" an object is. We can use the ratio of the minimum second moment to the maximum second moment as a measure of the roundedness of the object. If we have a somewhat elongated object such as this one, it will have a value for roundedness that is less that one because E_{min} is smaller than E_{max}.



Here are three example objects and their geometric properties computed using the moments discussed above. In each case, we applied a threshold to the gray-level image (first column) to obtain the binary image (second column). In the third column, the center (black dot) and the axis of minimum second moment (line) are overlaid on the binary image. Note that the disk does not have an axis because any axis through the center will yield the same second moment. In terms of roundedness, the first object is least rounded, the second slightly more rounded, and the third perfectly rounded.

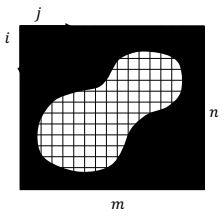


Thus far, we have shown how to take a binary image of an object, compute its area, location, and its maximum and minimum moments. The area and the two moments are useful features because they are not affected by translation and rotation of the object. We can use these properties to distinguish between a set of objects. After an object is recognized, the position and the orientation are used to enable a robot to pick up the object.

Now let us look at discrete binary images. Each cell is a pixel in the grid seen here, with a value of either 1 or 0. b_{ij} is the value of the binary image at the pixel in row *i* and column *j*. Let us assume that the area of each pixel is one. Then, the area of the object *A* is just the sum of all pixel values in the image. The expressions for computing the area (the zeroth moment) and the coordinates of the center (the first moments) in the case of a discrete image are shown here.

Discrete Binary Images

b_{ij}: Value at cell (pixel) in row i and column j.
Assume pixel area = 1.



Area: $A = \sum_{i=1}^{n}\sum_{j=1}^{m} b_{ij}$

Position: Center of Area (First Moment)

$$\bar{x} = \frac{1}{A}\sum_{i=1}^{n}\sum_{j=1}^{m} ib_{ij} \qquad \bar{y} = \frac{1}{A}\sum_{i=1}^{n}\sum_{j=1}^{m} jb_{ij}$$

20

We can also compute the second moments easily, without changing the coordinate system to the center of the object. We will explain shortly why this is relevant. Let us assume that the origin of the coordinate system is one of the corners of the image. We can then find the second moments *a'*, *b'* and *c'* using these expressions here, where all locations are measured with respect to the origin of the image, and not the center of the object. Using the moments *a'*, *b'* and *c'*, the center of the object, and its area, we can compute the moments *a*, *b*, and *c* with respect to the center of the object. A hint for how this can be done is given at the bottom of the slide.

Discrete Binary Images

Second Moments:

$$a' = \sum_{i=1}^{n}\sum_{j=1}^{m} i^2 b_{ij} \qquad b' = 2\sum_{i=1}^{n}\sum_{j=1}^{m} ijb_{ij} \qquad c' = \sum_{i=1}^{n}\sum_{j=1}^{m} j^2 b_{ij}$$

Note: a', b', c' are second moments w.r.t origin.
a, b, c (w.r.t. center) can be found from a', b', c', $\bar{x}$, $\bar{y}$, A

Hint: Expand $a = \sum_{i=1}^{n}\sum_{j=1}^{m}(i-\bar{x})^2 b_{ij}$ and represent in terms of a', $\bar{x}$, A.

21

Now, why did we not first find the center of the object and then compute the moments with respect to the center? Imagine that an image has been recorded by the image sensor. As the image is being read out from the sensor, pixel by pixel, the area *A*, center ($\bar{x}$, $\bar{y}$), and second moments *a'*, *b'* and *c'* can all be updated. Once the image is fully read out, the second moments *a*, *b*, and *c* with respect to the center of the object are easily computed. In short, all the relevant properties of the object are obtained during the read-out of the image.

Segmenting Binary Images

Shree K. Nayar
Columbia University

Topic: Binary Images, Module: Imaging
First Principles of Computer Vision

22

Multiple Objects

Need to Segment image into separate Components

Non-Trivial!

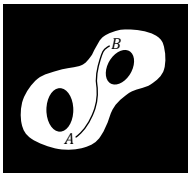
23

Next, we will discuss segmenting binary images into regions that correspond to different objects. When we discussed geometric properties, we made the assumption that there was only one object in the image. Since an image can have multiple objects, we need to label each object with a unique label before we compute its geometric properties. This is known as segmentation. While this appears like a trivial task, it turns out to be somewhat nuanced. We will assume that in our setting the objects do not overlap or even touch each other — that is, each object is surrounded by the background.

First, let us define what we mean by an object in a binary image. Two points, *A* and *B*, are considered to be "connected" if there exists a path from A to B along which the characteristic function of the binary image is constant. A connected component in a binary image is a maximal set of such connected points.

Connected Component

Maximal Set of Connected Points



A and B are connected if path exists between A and B along which $b(x, y)$ is constant.

24

Keeping this notion of a connected component in mind, let us develop a simple algorithm for segmenting a binary image, called region growing. By region, in this context, we mean an object. The algorithm starts by scanning the image in a raster scan fashion (each row is scanned from left to right and the rows are scanned from top to bottom) until it locates an unlabeled point with *b*=1. If such a point is not found, the algorithm terminates. If it is found, the point is assigned a new label (essentially a number) and becomes the seed point for a new region. This label is then assigned to all the neighbors of the seed point that have *b*=1, to the neighbors of those neighbors, and so on. This iterative process grows a region with the same label until there are no more unlabeled neighbors with *b*=1. When it reaches that point, the algorithm returns to the first step, which is to continue scanning for the next unlabeled seed point with *b*=1.

Connected Component Labeling

Region Growing Algorithm

(a) Find Unlabeled "Seed" point with $b = 1$.
If not found, Terminate.

(b) Assign New Label to seed point

(c) Assign Same Label to its Neighbors with $b = 1$

(d) Assign Same Label to Neighbors of Neighbors with $b = 1$. Repeat until no more Unlabeled Neighbors with b=1.

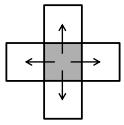
(e) Go to (a)

25

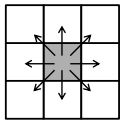
Before we proceed, let us take a closer look at what we mean by the term "neighbor." Here are two definitions for a neighbor. The first is based on 4-Connectedness (left), in which a pixel's neighbors are the four pixels to the left, right, top, and bottom of it. The other is 8-Connectedness (right), which includes the diagonal pixels. It turns out that neither of these definitions is perfect.

What do we mean by Neighbors?

Connectedness



4-Connectedness
4-C

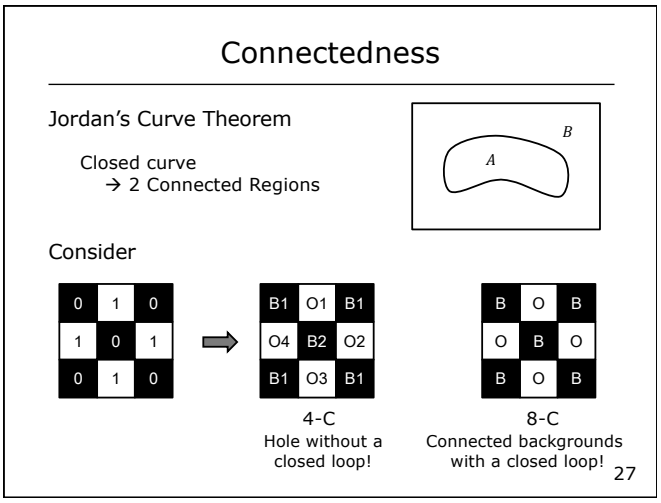


8-Connectedness
8-C

Neither is Perfect!

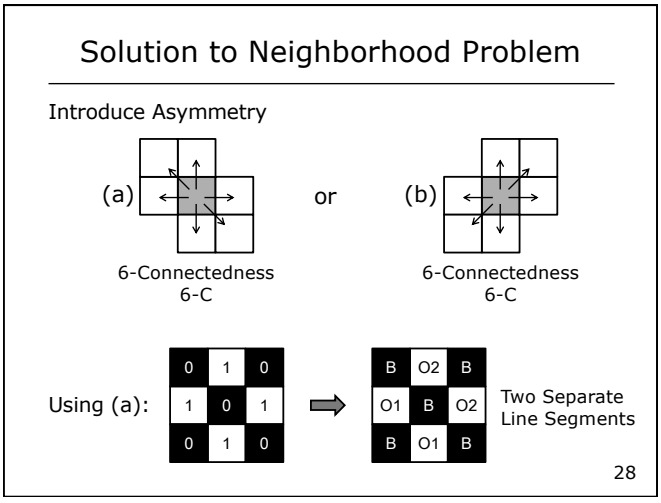
26

In order to understand the problems with these neighborhood definitions, let us consider Jordan's Curve Theorem. This theorem states that a closed curve must divide a region up into two connected regions. In the image in the top right corner, A and B are separate, fully connected regions that are not connected to each other. Both 4-Connectedness and 8-Connectedness violate Jordan's Curve Theorem.



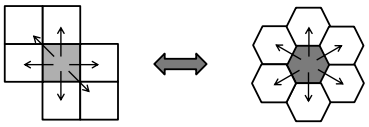
Consider the small binary image on the bottom left. We assume that the 0s in the outer ring of the image are connected to each other as the image shown here sits within a larger image with only 0s. In the case of 4-Connectedness (middle), none of the 1s are connected to each other because this definition neglects the diagonals. As a result, we obtain four separate objects (O1-O4). However, we do end up with two backgrounds, B1 and B2. So, we end up with four disconnected objects and two disconnected backgrounds, which violates Jordan's theorem. With 8-Connectedness (right), the four 1s are grouped into one object with the shape of a ring. However, in this case, the background inside the ring is connected to the background outside it, which again violates Jordan's theorem.

In order to address this problem, we introduce asymmetry into the definition of a neighbor. This is called 6-Connectedness. Shown on the top are two such asymmetric definitions of neighborhoods. Note that in each of these definitions, two diagonal pixels are dropped from the neighborhood. Returning to our example image patch, we now obtain two line segments and a single connected background. This outcome is consistent with Jordan's Curve Theorem.



By introducing the asymmetry above, we are, in effect, causing a square grid of pixels to behave like a hexagonal grid. On a hexagonal grid, each pixel has 6 well-defined neighbors. Unfortunately, today's commercially available image sensors can only capture images on square grids. So, our asymmetric definition for connectedness helps us segment binary images without violating Jordan's theorem.

Hexagonal Tessellation

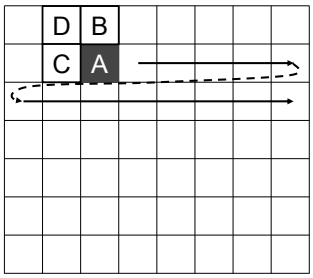


Above asymmetry makes a Square Grid behave like a Hexagonal Grid

29

Previously, we had discussed the region-growing algorithm for segmentation. Now we present an algorithm that is more elegant and efficient. Once again, we raster scan the image (left to right and top to bottom, as shown). We wish to label the pixel A. We will do this by only examining the labels of pixels B, C, and D. Since we are raster scanning the image, when we arrive at pixel A, we already have labeled pixels B, C, and D. We refer to this algorithm as sequential labeling. It is worth noting that, by using only the top, diagonal, and left neighbors to label a pixel, we are implicitly introducing the asymmetry needed to satisfy Jordan's theorem.

Sequential Labeling Algorithm

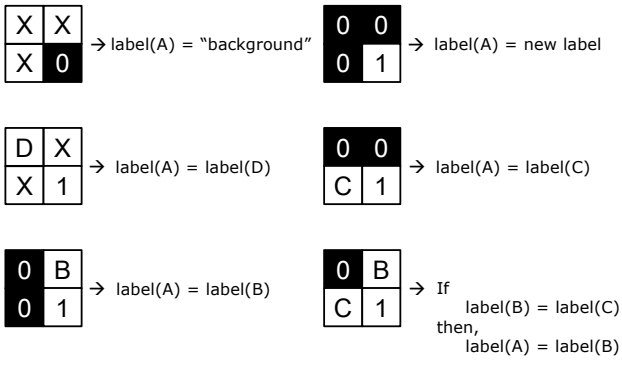


We want to label A.
B, C, D are already labeled.

31

Here is how sequential labeling works. If A is 0, we will call it part of the background, irrespective of what the labels of pixels B, C, and D are. If A is 1 and the three neighbors are 0, we give it a new label. If A is a 1 and D is labeled, we will give A the label of D, irrespective of the values of B and C. If A is 1, B and D are 0, and C is labeled, then A is given the label of C. Similarly, if C and D are 0, but B is labeled, then A is given the label of B. Now consider the case where A is 1, B and C have been labeled, and D is a background. If B and C have the same label, then A is given that label.

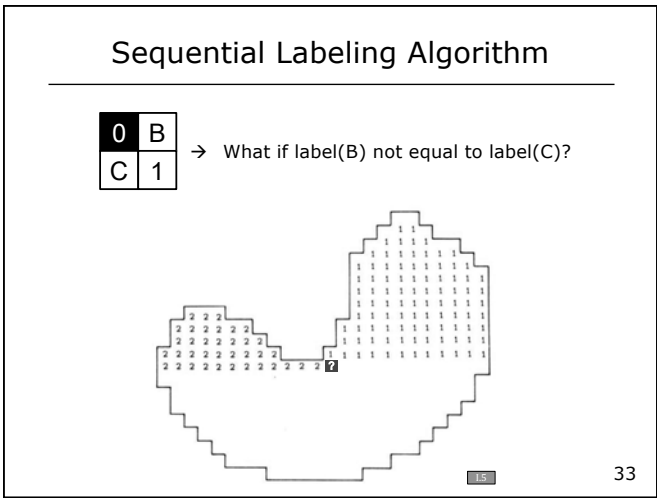
Sequential Labeling Algorithm



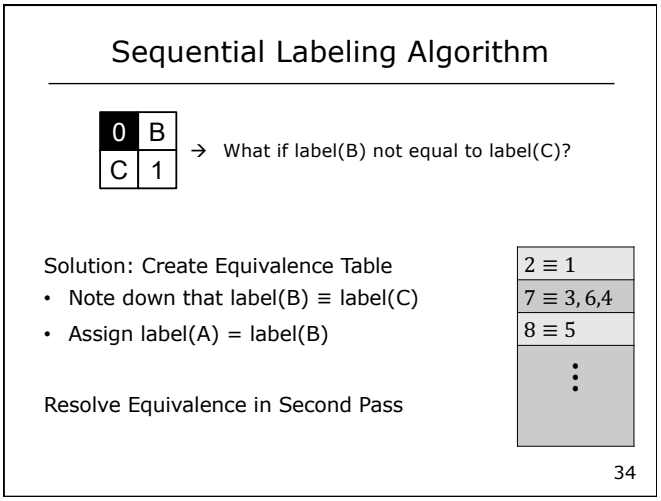
X: Value does not matter (Can be 0 or 1)

32

However, what do we do if D belongs to the background, B and C are labeled, but they have different labels? For the object shown here, this exact case arises for the pixel shown as a black box. Here, pixel D belongs to the background (it has the value 0) pixel B has the label 1 and pixel C has the label 2.



In this case, we assign pixel A the label of either B or C and simply make note of the fact that the label of B is "equivalent" to the label of C. This information is stored in an equivalence table. This table can be made compact by representing each set of equivalent labels by a single label. After the image has been fully scanned and labeled, we simply do a second raster scan of the image where the equivalences are resolved. In summary, with just two raster scans of the image, it is fully segmented into distinct regions, or objects. Now the geometric properties (moments) we defined previously can be computed for each of the objects in the image in order to recognize it and find its position and orientation.



Now let us examine ways in which a binary image can be iteratively modified to extract useful information from it. For example, given a binary image of a full human body, one could extract the skeleton of the body by thinning the body in the image. The skeleton could be used to determine information about the body, such as the lengths of its bones or its pose. When performing such an operation, it is crucial that the overall structure and integrity of the binary image is not compromised.

Iterative Modification

Shree K. Nayar
Columbia University

Topic: Binary Images, Module: Imaging
First Principles of Computer Vision

35

When considering the structure of a binary object, an important concept is the Euler number. The Euler number is the number of bodies minus the number of holes. For the letter B here, the number of bodies is 1 while the number of holes is 2, giving it a Euler number of -1. The letter i has 2 bodies and no holes, so its Euler number is 2. The letter n has a Euler number of 1. The Euler number of the full image is the sum of the Euler numbers of all its non-overlapping regions. This property is key as it implies that if an operation is applied to a region of the image and it does not change the Euler number of the region, then the Euler number of the complete image will remain the same.

Euler Number (E)

No. of Bodies (B) – No. of Holes (H)

Letter B: $E_b = -1$
Letter i: $E_i = 2$
Letter n: $E_n = 1$

Image : $E = 4$

$$E_{image} = \sum E_{non-overlapping\ regions}$$

36

We now define the Euler differential as the change in the Euler number of an image due to an operation applied to it. For our discussion here, we will assume, for convenience, that the image is a hexagonal grid. The same arguments can be applied to a square grid by assuming that the region the operation is applied to is asymmetric for the reasons discussed earlier. In the small image region (a pixel and its immediate neighbors) shown here, if the center pixel is changed from 0 to 1, the Euler number also changes from 0 to 1. Thus, the Euler differential is 1 minus 0, which is 1.

Euler Differential (E^*)

Change in the Euler number of the image.

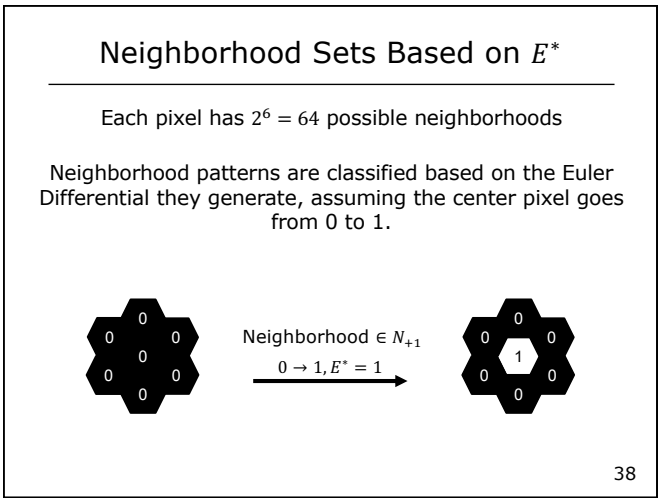
$E = 0$ $E = 1$

Euler Differential: $E^* = 1$

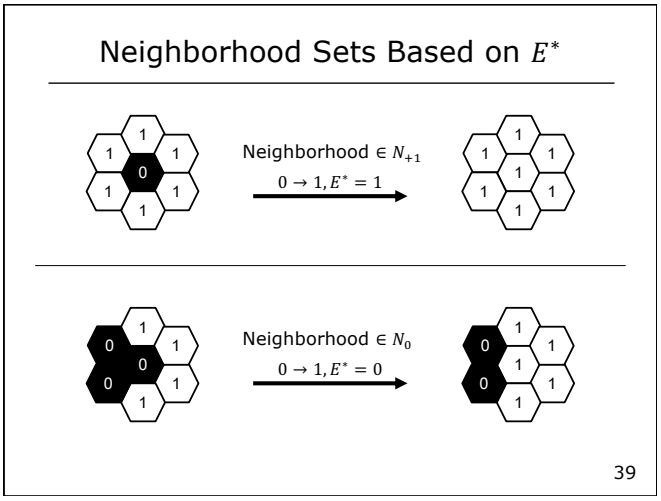
37

On a hexagonal grid, each pixel has 6 neighbors and hence 2^6 = 64 possible neighborhood patterns (patterns of 0s and 1s). These patterns can be classified based on the Euler differential they generate when the center pixel goes from a 0 to a 1. Whatever the Euler differential is, going from a 1 to a 0 would yield the negative of that number.

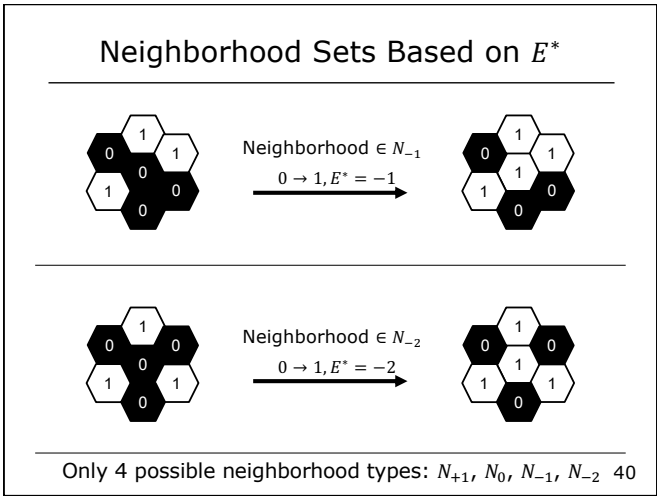
Let us take a look at an example. In this pattern, if the center pixel is changed from 0 to 1, the Euler differential is 1. We say that it belongs to the neighborhood class N_{+1}.



Here is another example that belongs to the neighborhood class N_{+1}. Below it we have a pattern that belongs to the neighborhood class N_0, which is a very important class because it represents conservative operators — these are neighborhood patterns for which the center pixel value can be modified without introducing any new bodies or holes in the pattern and hence in the larger image it belongs to.



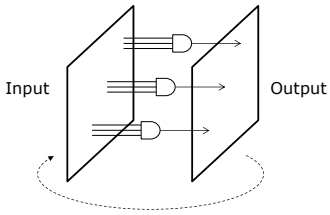
Here is a case where the Euler differential is -1. When the center pixel is changed from a 0 to a 1, two bodies are connected, causing the number of bodies to decrease by one. On the bottom, a pattern is shown that belongs to the neighborhood class N_{-2}. Upon examining all 64 neighborhood patterns, we find that there are only four possible neighborhood types: N_{+1}, N_0, N_{-1}, and N_{-2}.



This idea of neighborhood classes allows one to decide, based on the neighborhood of a pixel, what kinds of operations need to be applied to it to achieve a specific goal. In order to be completely safe, one would use an operation that is only applicable to the N_0 class of neighborhoods.

Iterative Neighborhood Operations

Incrementally apply neighborhood operations on images



Conservative Operations do not change the Euler number of the image.

41

Note that the operators we have discussed are local — pixels are modified solely based on their immediate neighbors. This has an important implication — the modifications can be applied to any number of pixels in parallel, as long as each modified pixel does not belong to the neighborhood of another pixel that is being modified at the same time. This condition is easy to satisfy. On a square grid, this can be achieved by dividing the pixels into three "fields" and applying the operation in parallel to all pixels in the same field. Since the operations are applied in parallel, they are highly efficient. In fact, the same operator can be applied repeatedly to an image until none of the pixels can be modified — hence the term iterative modification.

Let us develop a notation based on which we can define an entire class for iterative modification algorithms. A neighborhood set is first specified; we will call it the set *S,* which could be N_{+1}, N_0, N_{-1}, N_{-2}, or any combination of these. Let a pixel be denoted as (*i,j*) and let a_{ij} equal 1 if the neighborhood of pixel (*i,j*) belongs to *S*. Let us denote the current value of the pixel as b_{ij}. Based on these two values — a_{ij} and b_{ij} — a new value, $c_{ij,}$ will be assigned to the pixel. The two values, (a_{ij}, b_{ij}), could be (0,0), (0,1), (1,0), or (1,1). Corresponding to each one of these, c_{ij} will be assigned a binary output. Since there are four possible inputs, we have four outputs, and each one is a binary number. Therefore, we have $2^4 = 16$ possible ways in which the four $c_{ij,}$ outputs can be filled with binary numbers. Each one of these corresponds to an iterative modification algorithm.

Notation for Iterative Modification

Specify Neighborhood Set, S. Ex: S can be N_{+1}, N_0, N_{-1} or N_{-2} or a combination of these.

Consider pixel (i,j). Let:

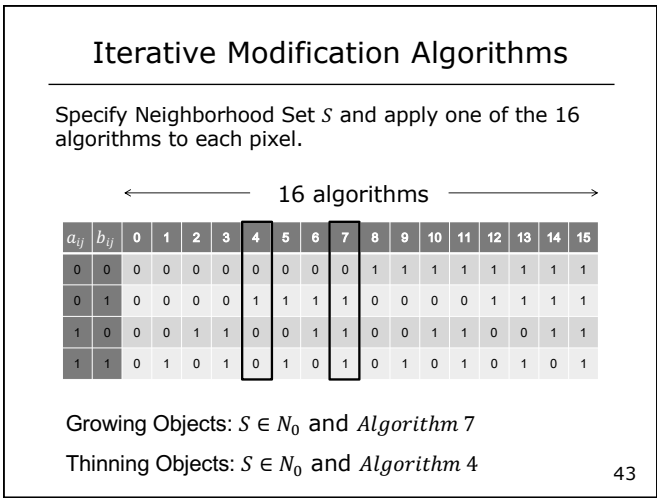
- $a_{ij} = 1$ if Neighborhood of $(i,j) \in S$ else 0
- b_{ij} = current value of pixel (i,j)
- c_{ij} = new value of pixel (i,j)

a_{ij}	b_{ij}		c_{ij}
0	0		?
0	1	⇨	?
1	0		?
1	1		?

$2^4 = 16$ algorithms

42

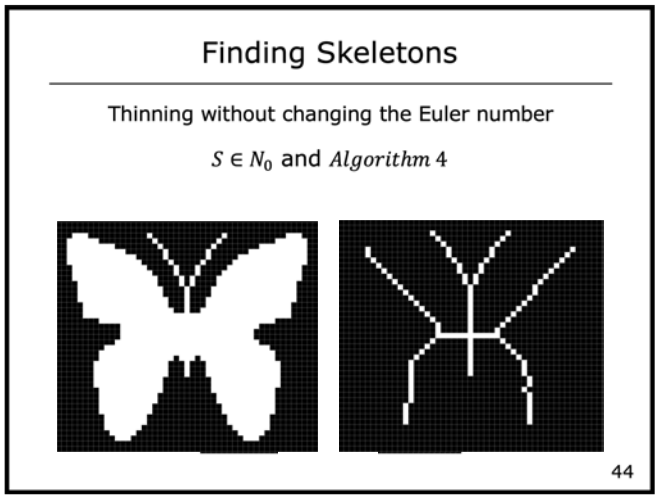
Shown here are all of the 16 algorithms for iterative modification illustrated as a table. Some of these are not particularly interesting because they modify the image in a way that is more akin to some sort of visual effect. These are not useful in the context of computer vision.

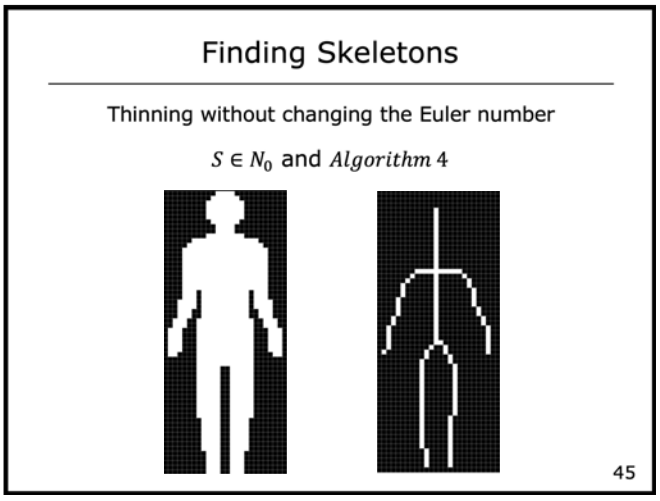


a_{ij}	b_{ij}	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

A few of the algorithms, however, are useful and widely used. Let us assume that the set *S* we are interested in is N_0. This means that we wish to be conservative and do not want to introduce any new objects or take away any existing objects. Now consider algorithm 7. In this case, 0s will be changed to 1s whenever possible, which means it will grow (or dilate) the object. In contrast, algorithm 4 does the opposite, meaning it will thin the object. Since we chose $S = N_0$, both dilation and thinning can be applied knowing that the objects in the image will not fuse together or break up into fragments.

Here we see the skeletons obtained by iteratively applying the thinning algorithm to binary images of a butterfly and a human body.





References and Credits

Shree K. Nayar
Columbia University

Topic: Binary Images, Module: Imaging
First Principles of Computer Vision

46

References: Textbooks

Robot Vision
Horn, B. K. P., MIT Press

47

Image Credits

I.1 https://pixabay.com/en/board-computer-chip-data-processing-564813. Public Domain.

I.2 Licensed under CC BY SA 3.0.

I.3 http://www.clker.com/clipart-hand-x-ray.html. Public Domain.

I.4 https://pixabay.com/en/board-computer-chip-data-processing-564813. Public Domain.

I.5 Robot Vision. B.K.P. Horn. MIT Press. McGraw-Hill, LLC. Used with permission.

I.6 Robot Vision. B.K.P. Horn. MIT Press. McGraw-Hill, LLC. Used with permission.

48

Acknowledgements: Thanks to Nisha Aggarwal and Jenna Everard for their help with transcription, editing and proofreading.

References

[Horn 1986] Robot Vision, Horn, B. K. P., MIT Press, 1986.

[Nayar 2022B] Image Formation, Nayar, S. K., Monograph FPCV-1-1, First Principles of Computer Vision, Columbia University, New York, February 2022.

[Nayar 2022C] Image Sensing, Nayar, S. K., Monograph FPCV-1-2, First Principles of Computer Vision, Columbia University, New York, February 2022.

[Nayar 2022E] Image Processing I, Nayar, S. K., Monograph FPCV-1-4, First Principles of Computer Vision, Columbia University, New York, March 2022.

[Nayar 2022H] Boundary Detection, Nayar, S. K., Monograph FPCV-2-2, First Principles of Computer Vision, Columbia University, New York, June 2022.

[Nayar 2025M] Image Segmentation, Nayar, S. K., Monograph FPCV-5-2, First Principles of Computer Vision, Columbia University, New York, May 2025.

References

[Horn 1986] Robot Vision, Horn, B. K. P., MIT Press, 1986.

[Nayar 2022B] Image Formation, Nayar, S. K., Monograph FPCV-1-1, First Principles of Computer Vision, Columbia University, New York, February 2022.

[Nayar 2022C] Image Sensing, Nayar, S. K., Monograph FPCV-1-2, First Principles of Computer Vision, Columbia University, New York, February 2022.

[Nayar 2022E] Image Processing I, Nayar, S. K., Monograph FPCV-1-4, First Principles of Computer Vision, Columbia University, New York, March 2022.

[Nayar 2022H] Boundary Detection, Nayar, S. K., Monograph FPCV-2-2, First Principles of Computer Vision, Columbia University, New York, June 2022.

[Nayar 2025M] Image Segmentation, Nayar, S. K., Monograph FPCV-5-2, First Principles of Computer Vision, Columbia University, New York, May 2025.