INTRODUCTION TO MORPHOLOGY AND IMAGE COMPRESSION

DR. BHAKTI PALKAR

5.1

Morphological operations:

1. Dilation

2. Erosion

3. Opening

4. Closing

5. Hit or Miss Transform

6. Boundary extraction

DILATION

The dilation process the structuring element is reflected and shifted from left to right and from top to bottom, at each shift; the process will look for **any overlapping** similar pixels between the structuring element and that of the binary image.
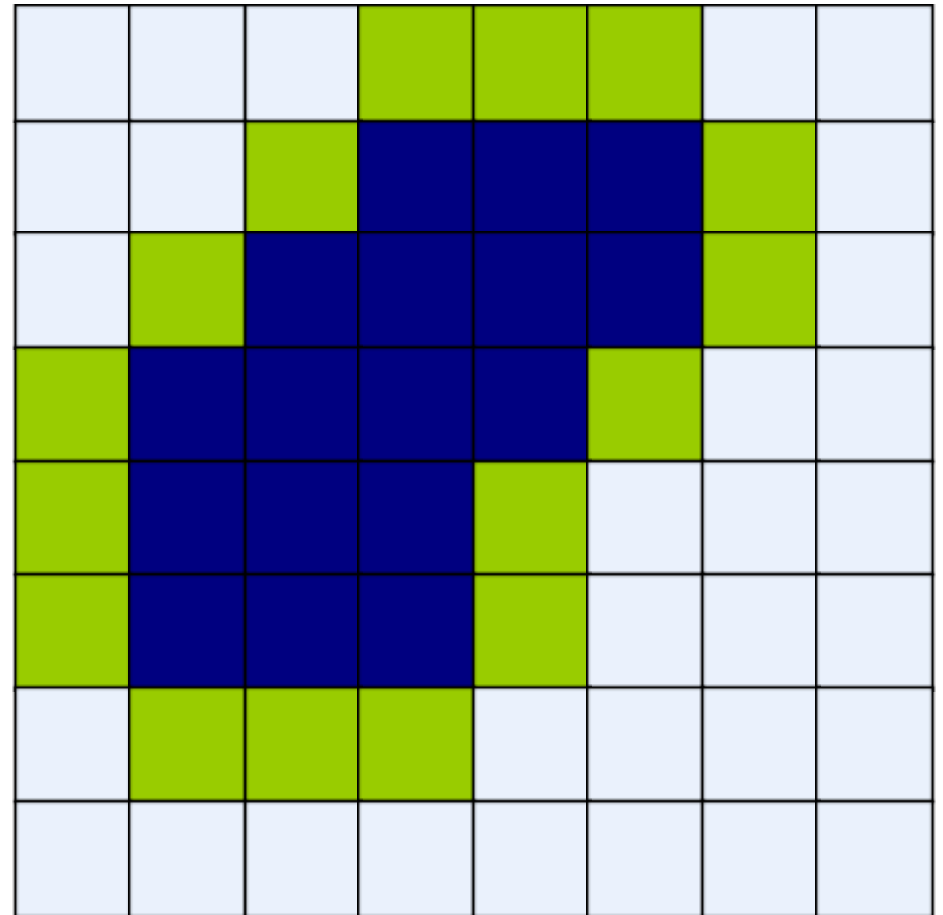
If there exists an overlapping then the pixels under the centre position of the structuring element will be turned to 1 or black.

$$A \oplus B = \left\{ x \mid (\hat{B})_x \cap A \neq \phi \right\},$$

Original Image
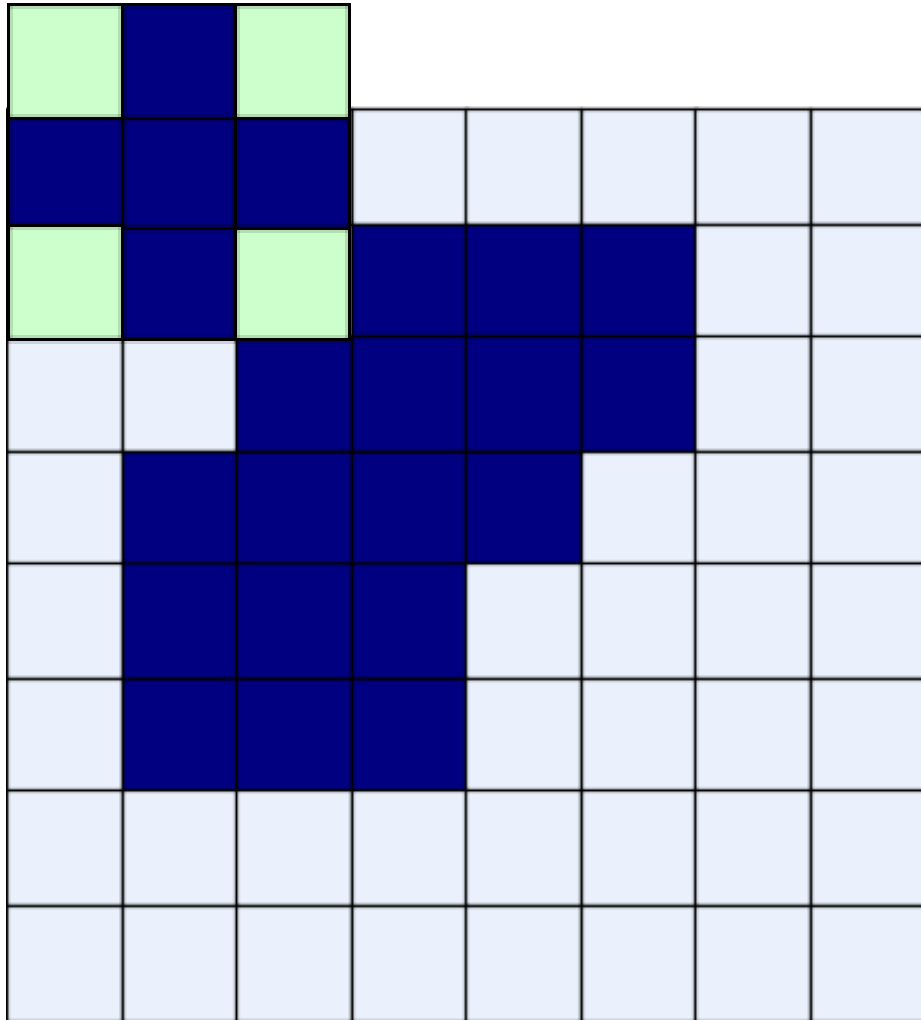
Processed Image With Dilated Pixels

Structuring Element

DILATION EXAMPLE

Original image
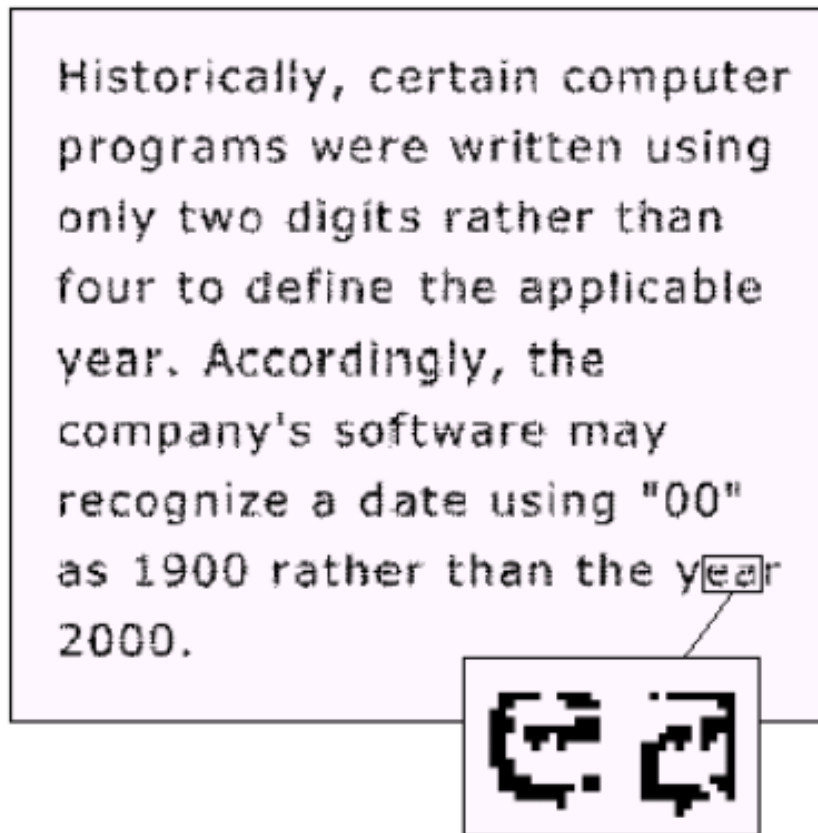
Dilation by 3*3 square structuring element
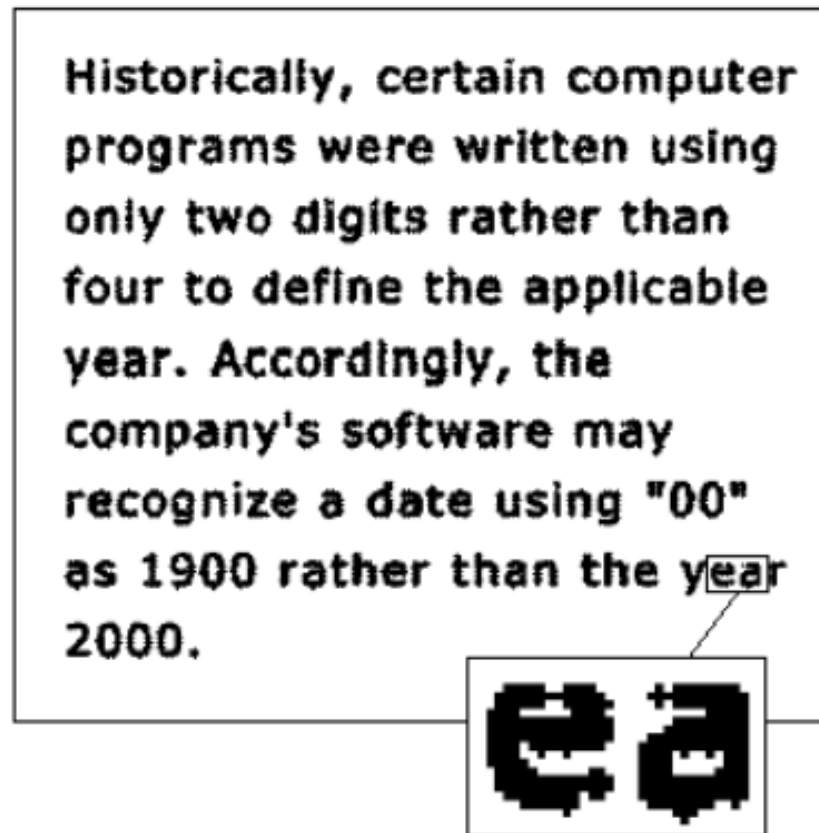
Dilation by 5*5 square structuring element

Watch out: In these examples a 1 refers to a black pixel!

DILATION EXAMPLE

Original image

Historically, certain computer programs were written using only two digits rather than four to define the applicable year. Accordingly, the company's software may recognize a date using "00" as 1900 rather than the year 2000.

After dilation

Historically, certain computer programs were written using only two digits rather than four to define the applicable year. Accordingly, the company's software may recognize a date using "00" as 1900 rather than the year 2000.

0	1	0
1	1	1
0	1	0

Structuring element

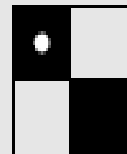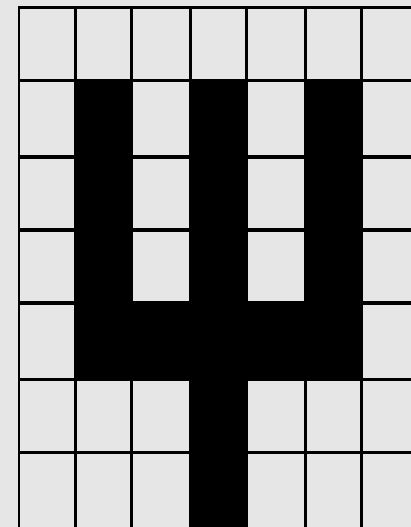# WHAT IS DILATION FOR?

Dilation can repair breaks

Dilation can repair intrusions

**Watch out**: Dilation enlarges objects

Example 10.1 *Using the input image and structuring element as given below, find the dilated version of the input image.*

Step 1 Compare the structuring element with the input image at the centre point. An overlapping component between the structuring element and the input image is marked as *. Hence the origin of the structuring element corresponding to the input pixel is turned to on.
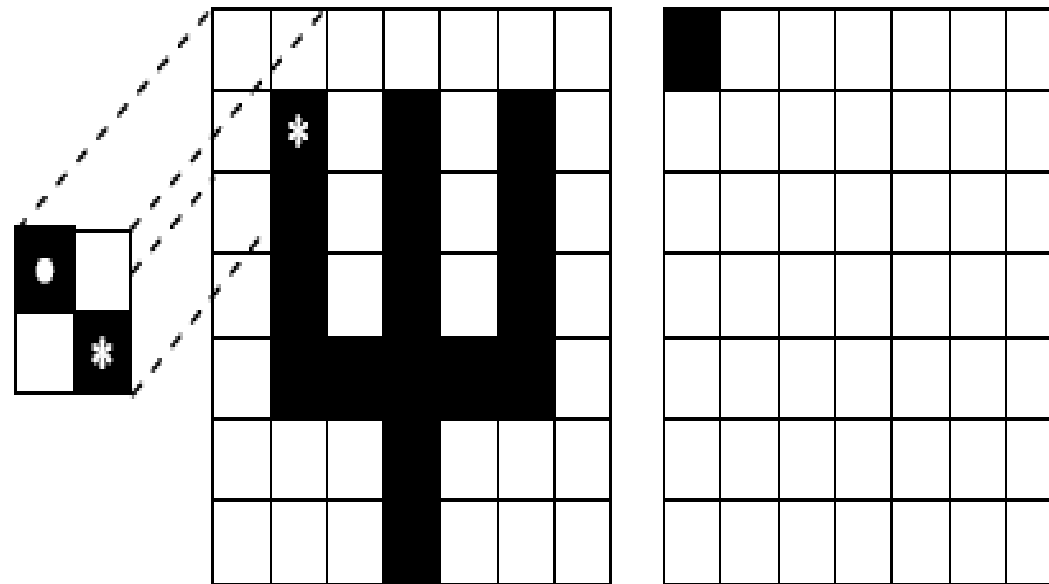
Step 3 Similar to the previous step, the structuring element is shifted towards the right side of the input image to find if any overlapping is present. In this case, overlapping is present, and so the corresponding input pixel is turned to on the resultant image and is shown as follows:
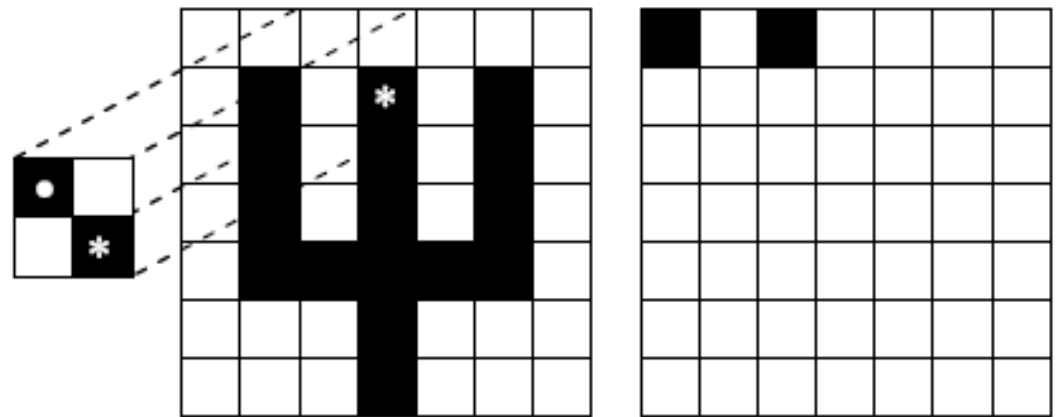
Step 4 Now the structuring element is again shifted to the right side of the input image to find any overlapping. There is an overlapping and the resultant image is shown below.
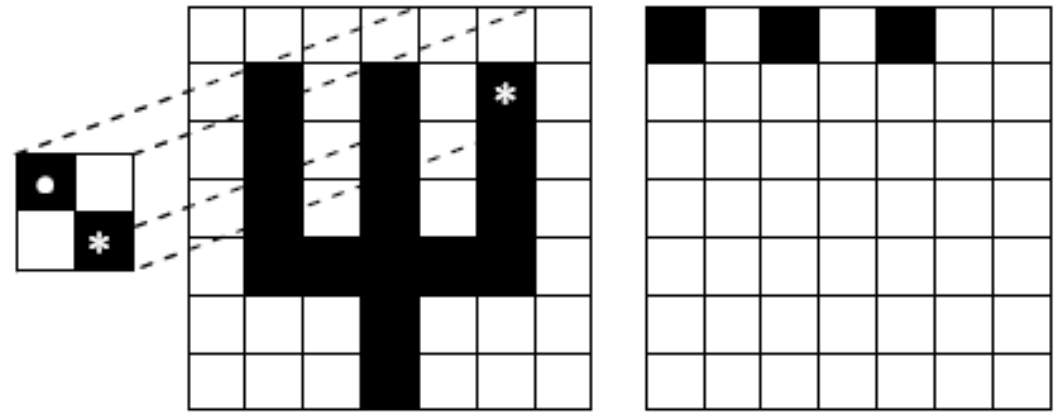
Step 6 The above process is repeated to get the final output image which is shown below.
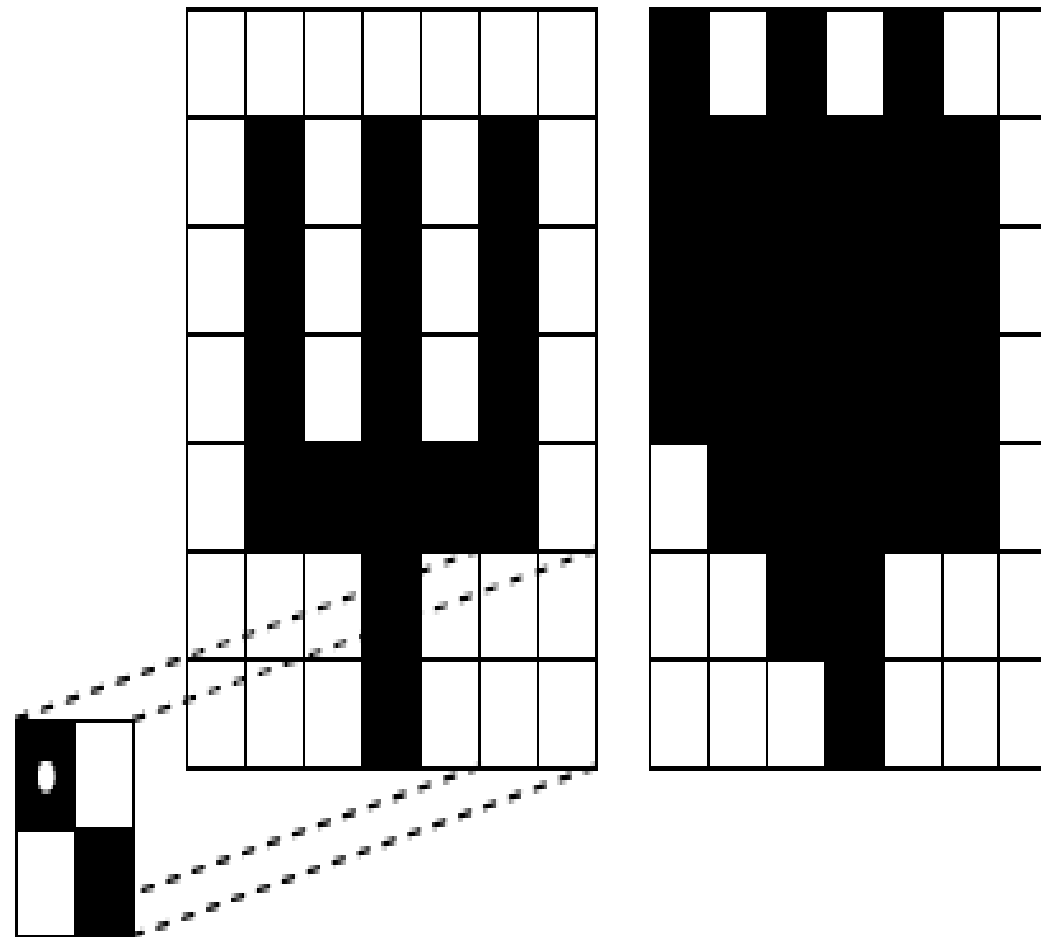
EROSION

Erosion is the counter-process of dilation. If dilation enlarges an image then erosion shrinks the image.

The process will look for whether there is a complete overlap with the structuring element or not.

If there is no complete overlapping then the centre pixel indicated by the centre of the structuring element will be set white or 0
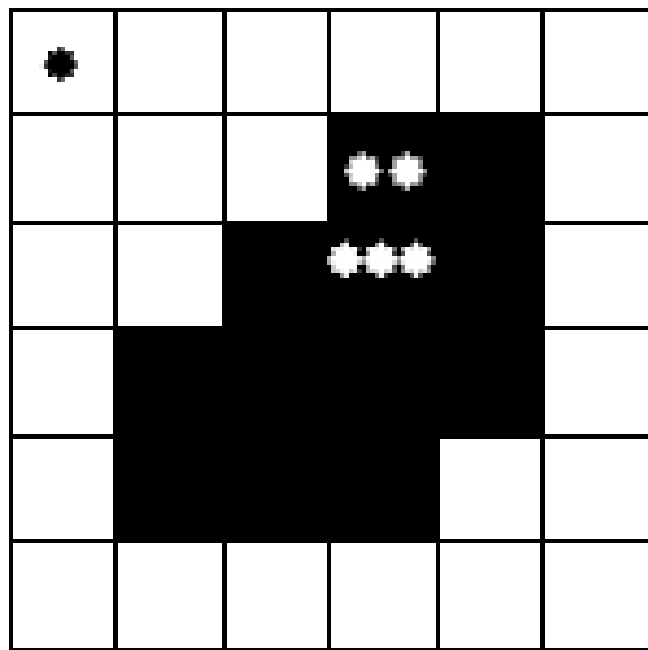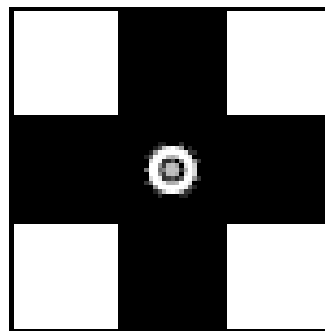
Input image   Structuring element   Eroded image

Fig. 10.15 *Erosion process*

Example 10.2 *The input image and structuring element are given below. Find the eroded version of the input image.*
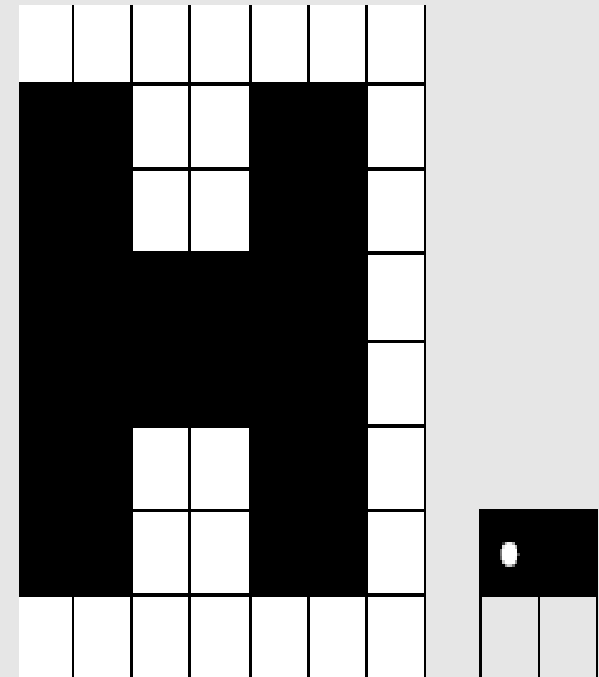
Solution

Step 1 The structuring element is compared with the input image. If all the pixels in the structuring element is overlapped with the input image then the origin corresponding to the input image pixel is turned ON. Otherwise that pixel is turned OFF. In our c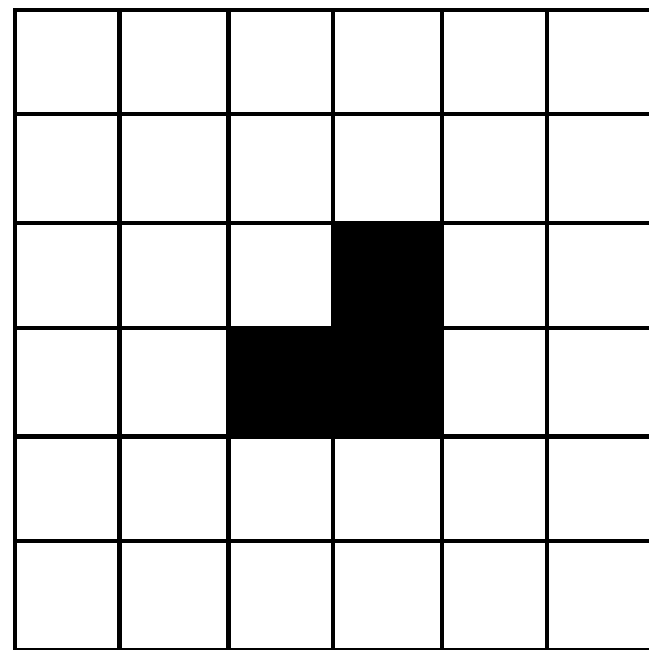ase, the structuring element is shifted from left to right. Initially, there are no pixels overlapping between the structuring element and input image, so the output image is shown below for first iteration. In the first row, the input image is not perfectly overlapped with the structuring element. Hence, all the pixels of the output image in the first row are OFF.
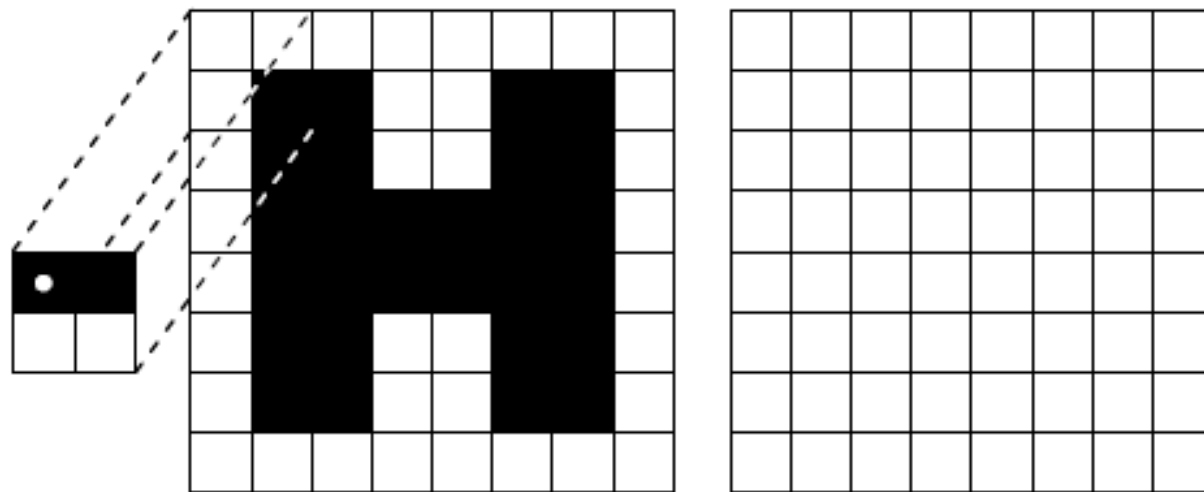
Step 2 Now the structuring element is mapped with the second row of the input image. Initially, there is no perfect matching; hence shift one pixel towards the right. After one shift towards the right, we find a perfect match, and hence the output image pixel is turned black which is illustrated below.
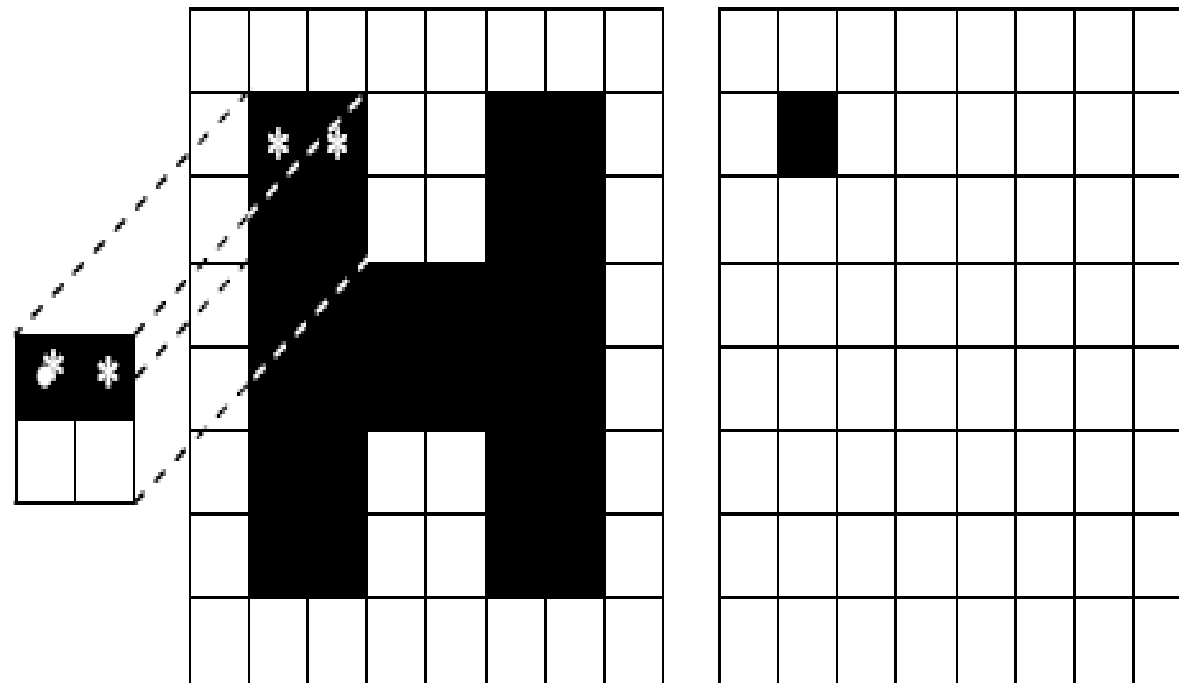
Step 3 The above process is repeated till the last pixel in the input image is taken into account. The eroded image after reaching the last pixel in the input image is shown below.
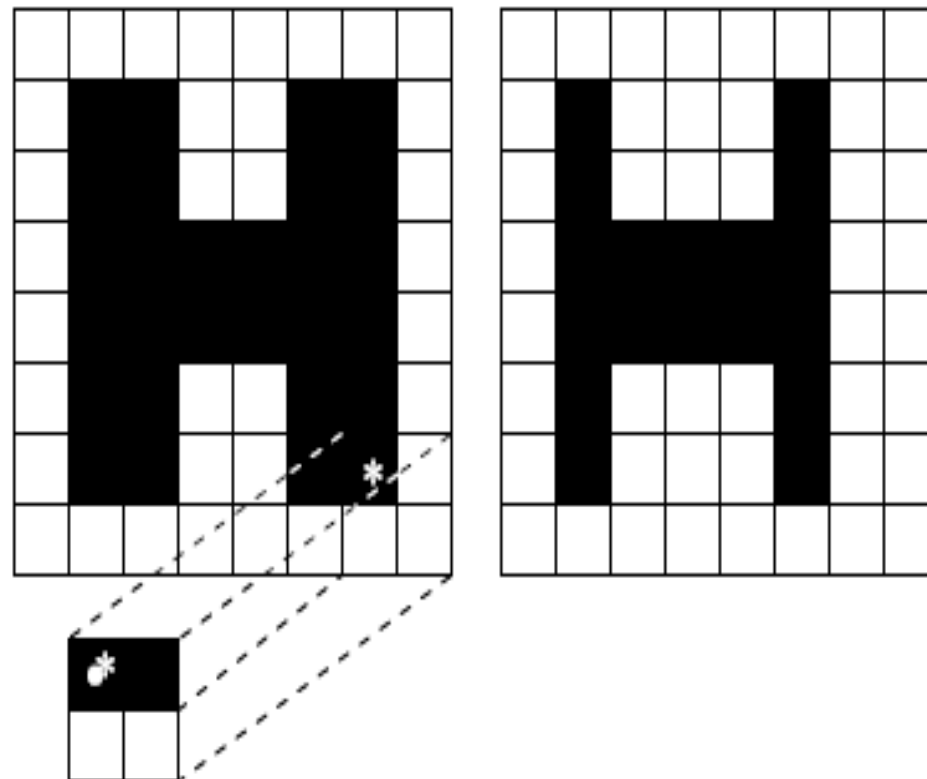
COMPOUND OPERATIONS

More interesting morphological operations can be performed by performing combinations of erosions and dilations

The most widely used of these *compound operations* are:

- Opening
- Closing

OPENING

The opening of image *f* by structuring element *s,* denoted *f ∘ s* is simply an <u>erosion followed by a dilation</u>

A	$A \ominus B$	$A \circ B = (A \ominus B) \oplus B$
Original shape	After erosion	After dilation (opening)
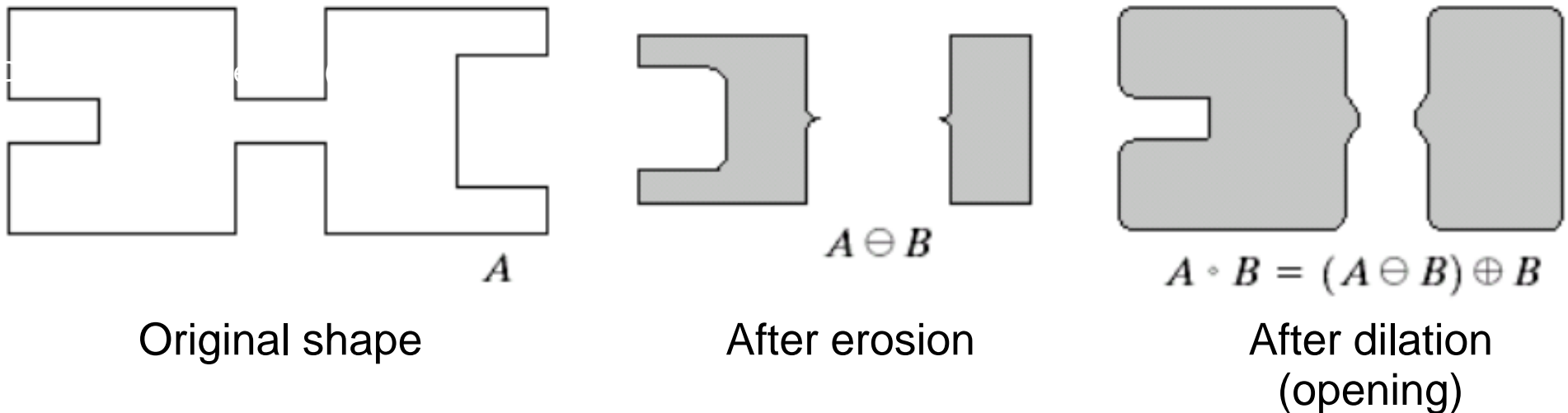
Note a disc shaped structuring element is used

OPENING EXAMPLE

Original Image
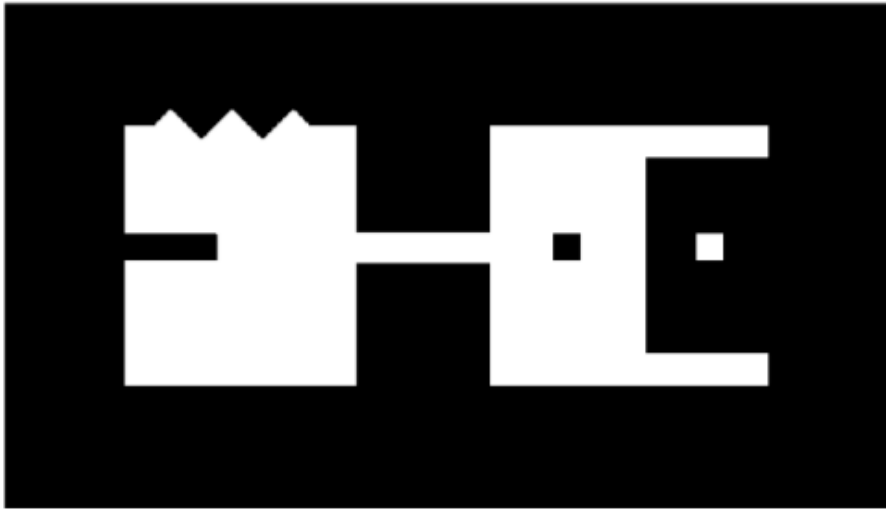
Image After Opening

Images taken from Gonzalez & Woods, Digital Image Processing (2002)

OPENING EXAMPLE

Original Image
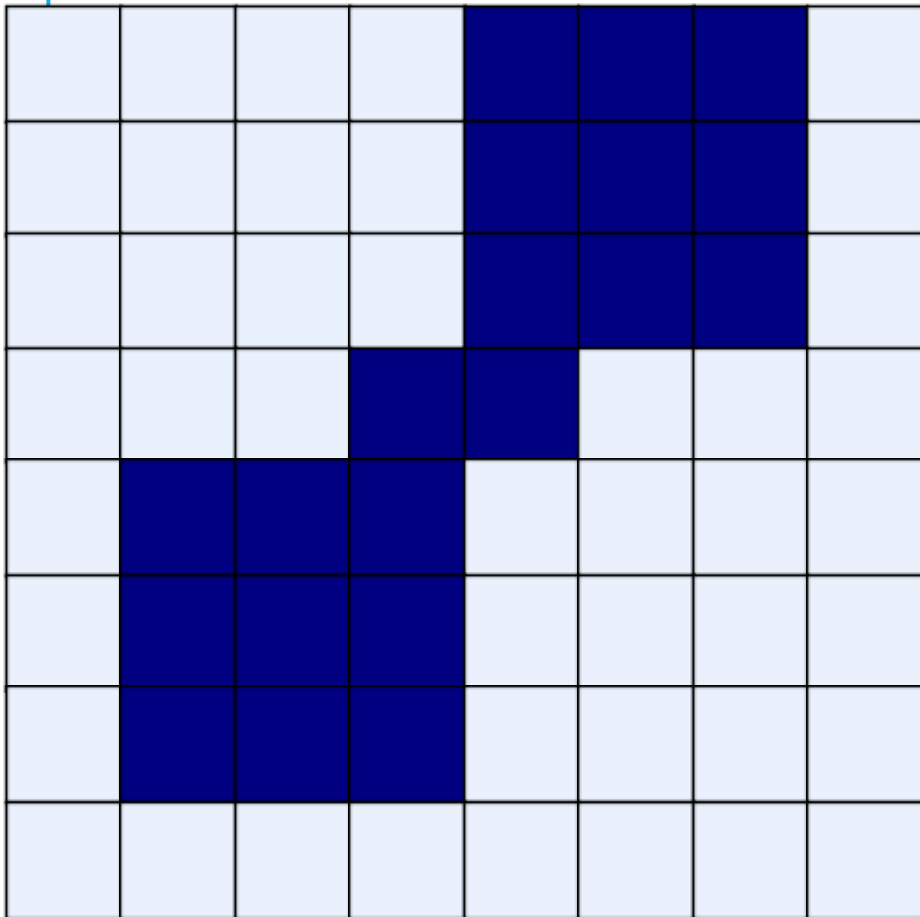
Processed Image
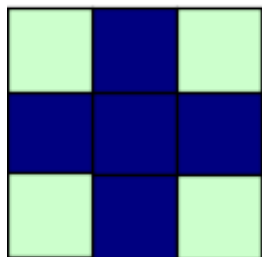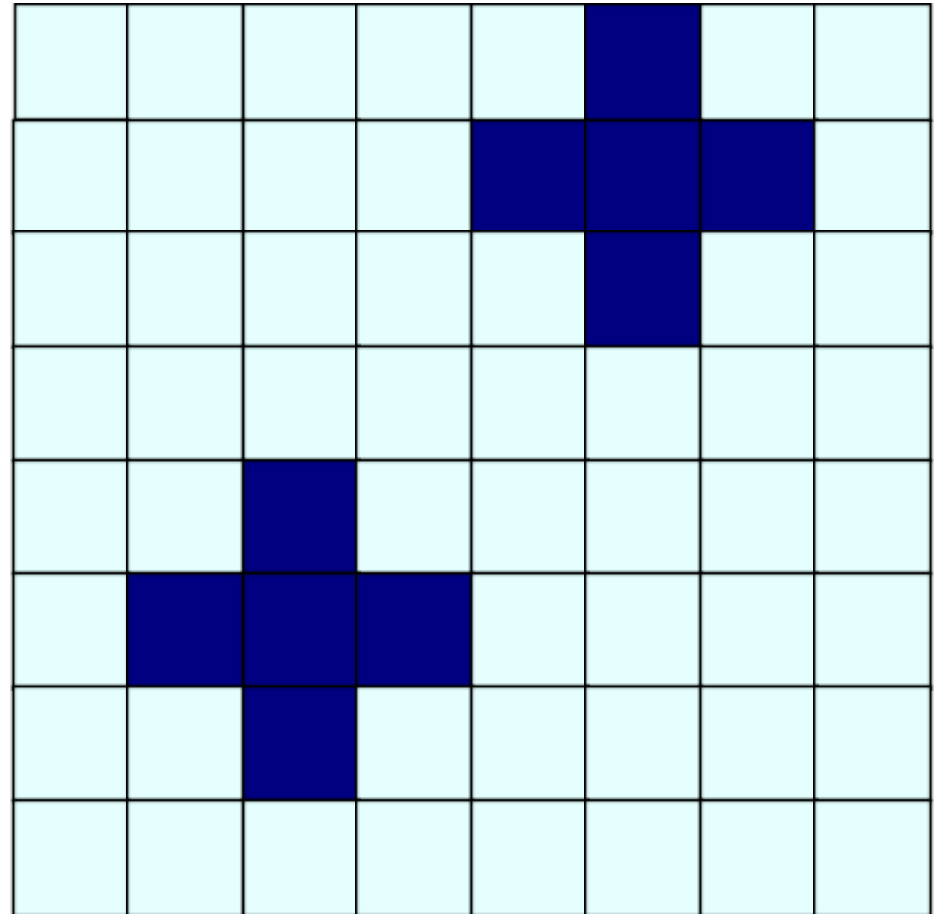
Structuring Element

CLOSING

The closing of image *f* by structuring element *s,* denoted *f • s* is simply a <u>dilation followed by an erosion</u>
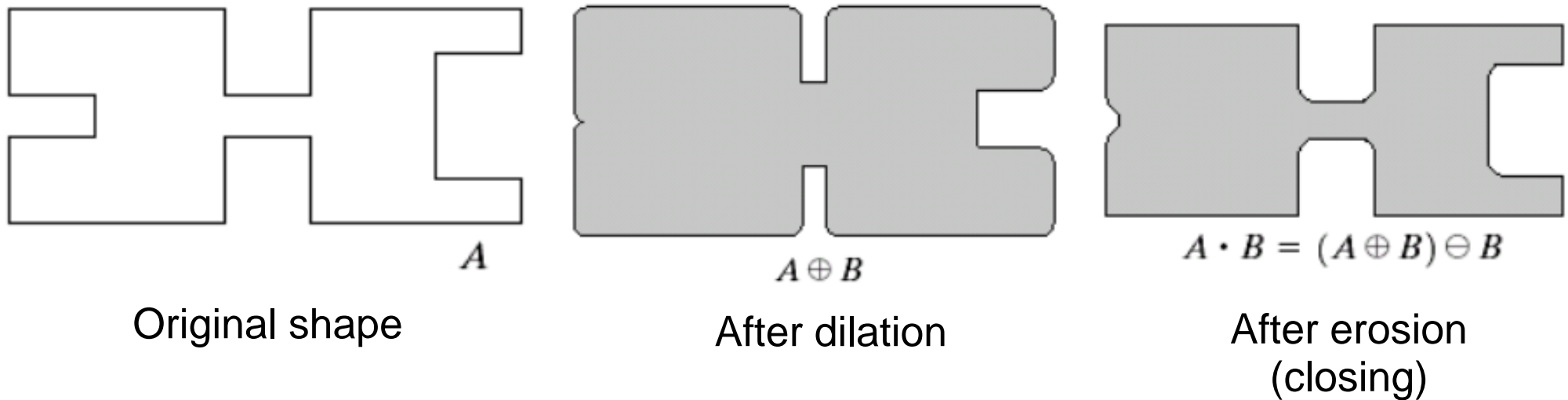
A	*A ⊕ B*	$A \cdot B = (A \oplus B) \ominus B$
Original shape	After dilation	After erosion (closing)

Note a disc shaped structuring element is used

Images taken from Gonzalez & Woods, Digital Image Processing (2002)

CLOSING EXAMPLE

Original Image

Image After Closing

Images taken from Gonzalez & Woods, Digital Image Processing (2002)
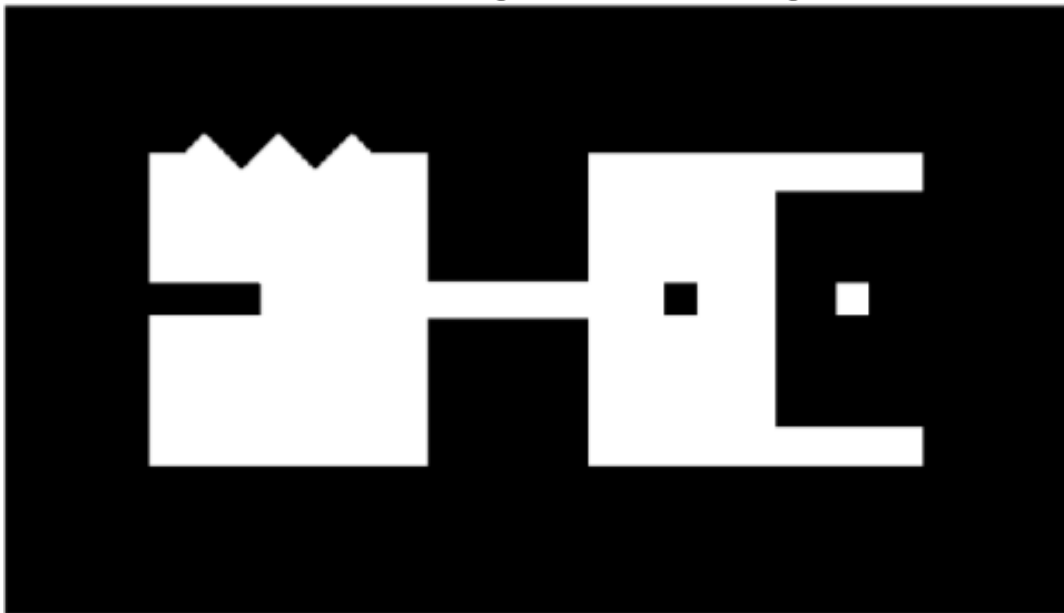
CLOSING EXAMPLE

Original Image
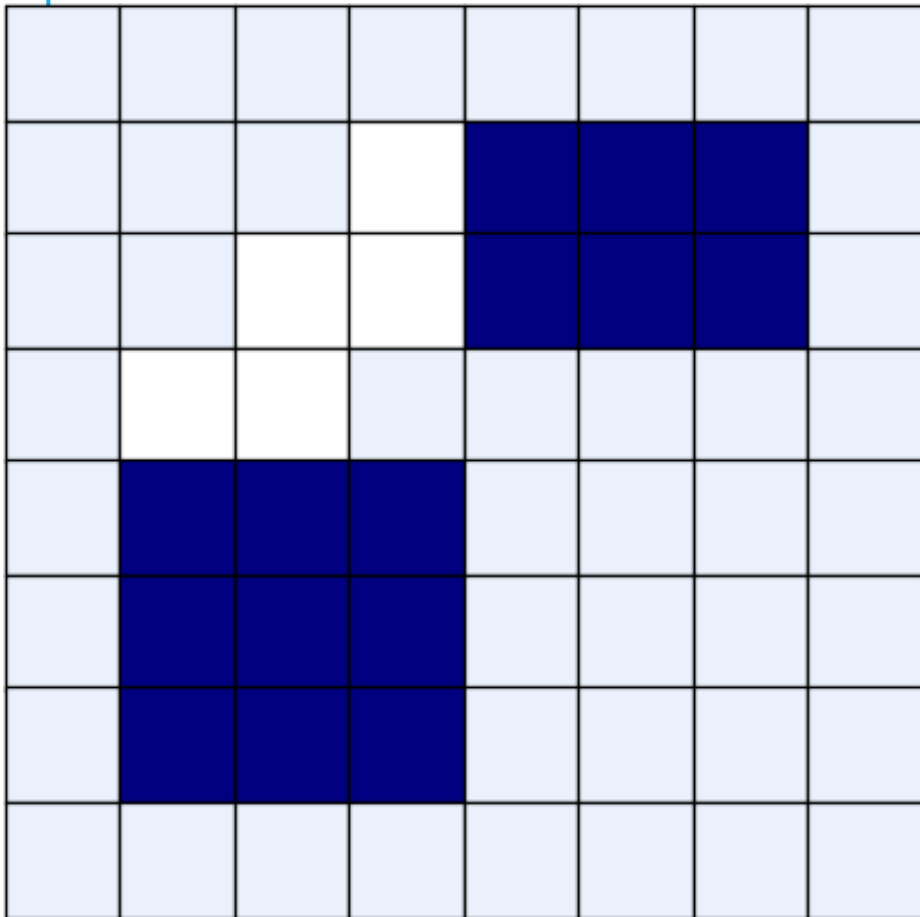
Processed Image
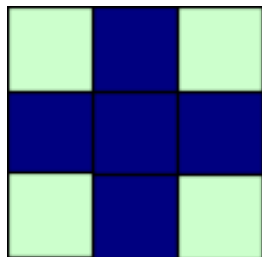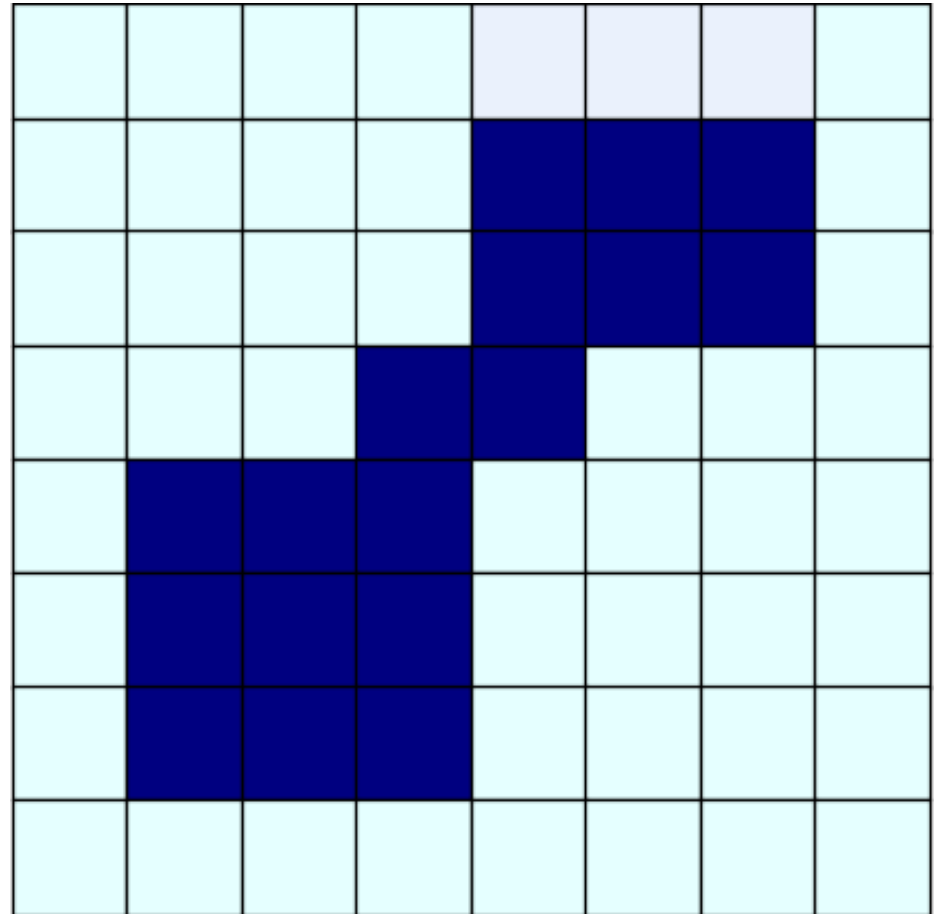
Structuring Element

MORPHOLOGICAL PROCESSING EXAMPLE

Images taken from Gonzalez & Woods, Digital Image Processing (2002)

MORPHOLOGICAL ALGORITHMS

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

- Boundary extraction
- Region filling
- Extraction of connected components
- Thinning/thickening
- Skeletonization
- Convex Hull
- Pruning

HIT AND MISS TRANSFORM

The hit-or-miss transform is a transformation which is used for template matching.

The transformation involves two template sets, B and (W−B), which are disjoint.

Template B is used to match the foreground image, while (W−B) is used to match the background of the image.

The hit-or-miss transformation is the intersection of the erosion of the foreground with B and the erosion of the background with (W−B).

The hit-or-miss transform is defined as

$$HM(X, B) = (X \ominus B) \cap \left(X^c \ominus (W - B)\right)$$

Example 10.4   *The input image and structuring elements are shown below. Find the hit or miss transformation for the input image.*
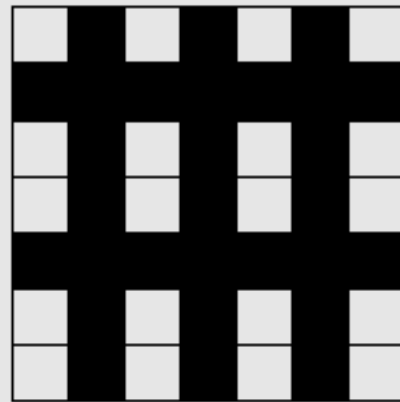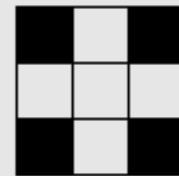
(a) Input image   (b) Structuring   (c) Structuring
                         element B element $W-B$
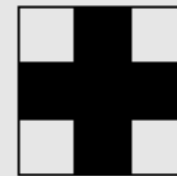
Solution

Step 1 From the definition of hit-or-miss transformation, we have to find the erosion of the input image with the structuring element B. The result of the erosion operation is shown below.
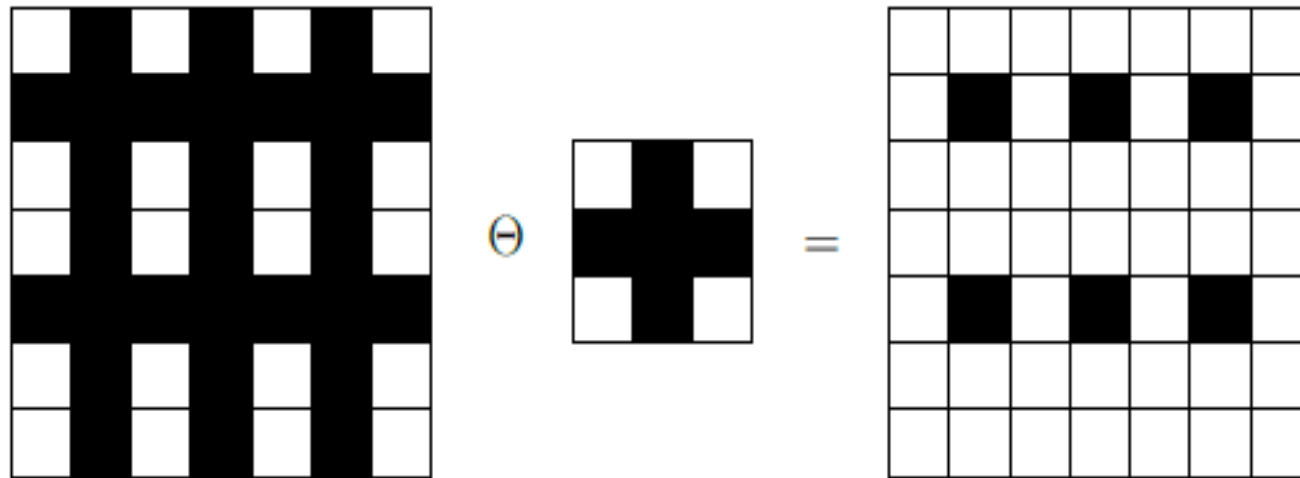
Step 2 Next, find the erosion of the complement of the input image with the structuring element $W-B$; we get the output image as shown below.
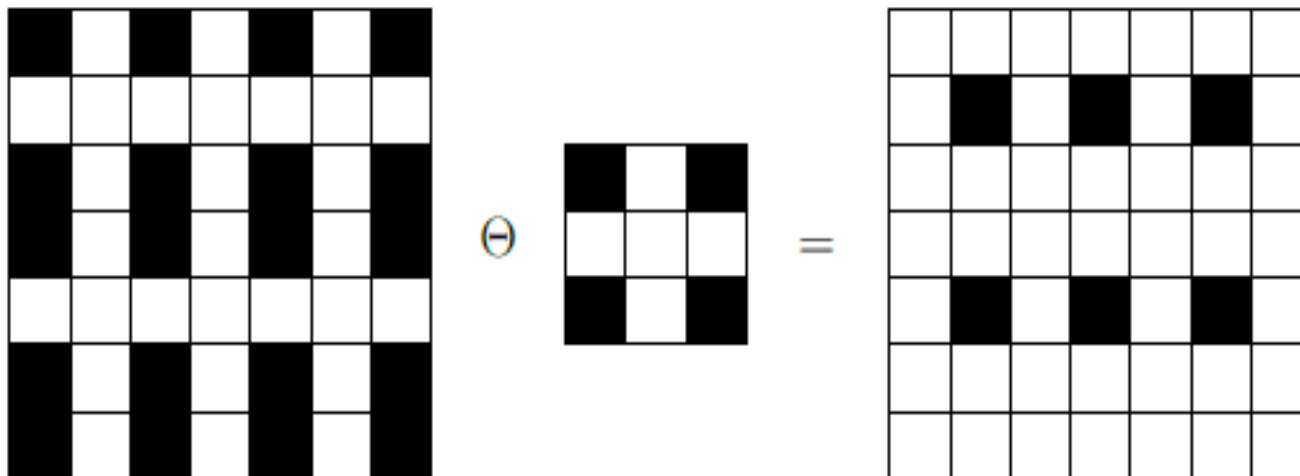
Step 3 From the result of steps 1 and 2, the hit-or-miss transformed input image is found from the intersection of the result of the image in steps 1 and 2. The resultant image is shown below.
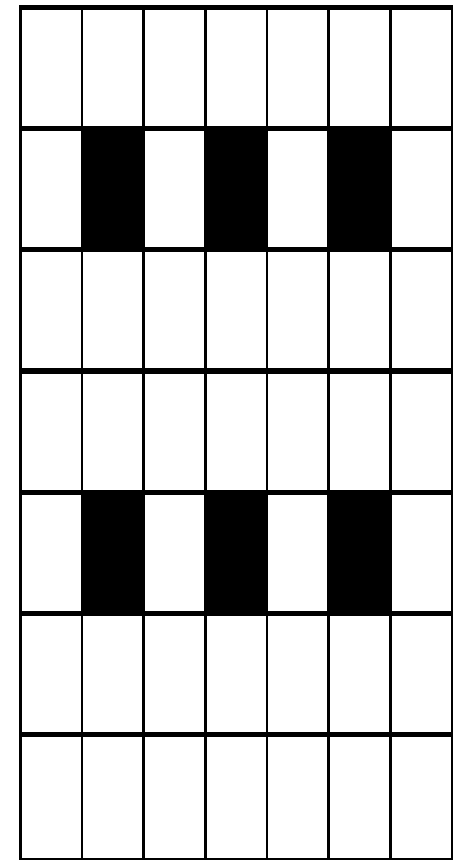
5.2

Introduction, redundancies: coding, inter-pixel, psycho-visual, compression ratio, fidelity criteria

Lossless compression techniques: Run length coding, Arithmetic coding, Huffman coding, Differential PCM

5.3

Lossy Compression techniques: Improved grey scale quantization, Vector quantization, Transform coding, JPEG.

IMAGE COMPRESSION

- Image compression means the reduction of the amount of data required to represent a digital image by removing the redundant data. It involves reducing the size of image data files, while retaining necessary information.

- Mathematically, this means transforming a 2D pixel array (i.e. image) into a statistically uncorrelated data set. The transformation is applied prior to storage or transmission of the image. At later time, the compressed image is decompressed to reconstruct the original (uncompressed) image or an approximation of it.

COMPRESSION RATIO

- The ratio of the original (uncompressed) image to the compressed image is referred to as the *Compression Ratio C_R*:

$$C_R = \frac{Uncompressed\ Image\ Size}{Compressed\ Image\ Size} = \frac{Usize}{Csize}$$

where

$$Usize = M \times N \times k$$

$$Csize = size\ of\ compressed\ image\ file\ stored\ in\ a\ disk$$

COMPRESSION RATIO

Example:

Consider an 8-bit image of 256×256 pixels. After compression, the image size is 6,554 bytes. Find the compression ratio.

Solution:

$Usize = (256 \times 256 \times 8) / 8 = 65,536$ bytes

$Compression\ Ratio = 65536 / 6554 = 9.999 \approx 10$ (also written 10:1)

This means that the original image has 10 bytes for every 1 byte in the compressed image.

REDUNDANCY AND COMPRESSION RATIO

Let n_1 and n_2 denote information carrying units in two data sets representing same information.

The relative data redundancy of 1st set n_1 is defined as $R_D = 1 - \dfrac{1}{C_R}$, where $C_R = \dfrac{n_1}{n_2}$ is called the compression ratio.

When $n_1 = n_2$, $C_R = 1$, $R_D = 0$ and there is no redundancy.

When $n_2 \ll n_1$, $C_R \to \infty$, $R_D \to 1$ showing highest redundancy and greatest compression.

When $n_2 \gg n_1$, $C_R \to 0$, $R_D \to -\infty$, showing no compression.

IMAGE DATA REDUNDANCIES

- Coding redundancy: occurs when the data used to represent the image are not utilized in an optimal manner. For example, we have an 8-bit image that allows 256 gray levels, but the actual image contains only 16 gray levels (i.e. only 4-bits are needed).

- Interpixel redundancy: occurs because adjacent pixels tend to be highly correlated. In most images, the gray levels do not change rapidly, but change gradually so that adjacent pixel values tend to be relatively close to each other in value.

IMAGE DATA REDUNDANCIES

- Psychovisual redundancy: means that some information is less important to the human visual system than other types of information. This information is said to be *psychovisually redundant* and can be eliminated without impairing the image quality.

Image compression is achieved when one or more of these redundancies are reduced or eliminated.

Coding redundancy:

We assume grey levels are random quantities.

The values of these levels are represented by suitable encoding them.

Let r_k in an interval $[0,1]$ represent the grey levels.

Let each r_k occur with probability $p_r(r_k) = \dfrac{n_k}{n}$, with $k = 0,1,2,\ldots L-1$ where L is

number of grey levels.

Let $l(r_k)$ bits be used to represent the value r_k

Then average number of bits required to represent each pixel

$$L_{avg} = \sum_{k=0}^{L-1} l(r_k) p_r(r_k)$$

And total bits in the image of dimension $M \times N$ is MNL_{avg}.

For natural m-bit binary code, $L_{avg} = m$

	r_k	$p_r(r_k)$	Code 1	$l_1(r_k)$	Code 2	$l_2(r_k)$
r_0	0	0.19	000	3	11	2
r_1	1/7	0.25	001	3	01	2
	2/7	0.21	010	3	10	2
	3/7	0.16	011	3	001	3
		0.08	100	3	0001	4
		0.06	101	3	00001	5
		0.03	110	3	000001	6
r_7	1	0.02	111	3	000000	6

$$L_{avg} = 2(0.19) + 2(0.25) + 2(0.21) + 3(0.16) + 4(0.08) + 5(0.06) + 6(0.03) + 6(0.02) = 2.7 bits$$

$$C_R = \frac{3}{2.7} = 1.1, \quad R_D = 1 - \frac{1}{1.11} = 0.099 \quad \text{The relative data redundancy}$$

- Here coding has been used to minimize the number of symbols using variable length coding, and code 1 has redundant code symbols. This is called removal of coding redundancy.
- It uses the fact that generally images have regular and predictable shape and area much large than the elements
- Hence some values are more probable than others, and they are coded with shorter length codes.

HUFFMAN CODING

Say, Image size: 10 x 10 (5 bit image)

Frequency:

a2 = 40   a6 = 30   a1 = 10   a4 = 10   a3 = 6   a5 = 4

P(a2) = 40/100 = 0.4

Symbols (like intensity levels)	Probabilities (sorted)	Source Reduction (do till two values are left) (Maintain in sorted order here as well)			
		1	2	3	4
a2	0.4	0.4	0.4	0.4	0.6
a6	0.3	0.3	0.3	0.3	0.4
a1	0.1	0.1	0.2	0.3	
a4	0.1	0.1	0.1		
a3	0.06	0.1			
a5	0.04				

Symbols (like intensity levels)	Probabilities (sorted)	Source Reduction (do till two values are left) (Maintain in sorted order here as well)			
		1	2	3	4
a2	0.4 1	0.4	0.4	0.4	0.6 0
a6	0.3 00	0.3	0.3	0.3 00	0.4 1
a1	0.1 011	0.1	0.2 010	0.3 01	
a4	0.1 0100	0.1 0100	0.1 011		
a3	0.06 01010	0.1 0101			
a5	0.04 01011				

Encoded String: 010100111100

Decoding   : a3 a1 a2 a2 a6

Parameters:

1. Average length of code

   Lavg = 0.4 * 1 + 0.3 * 2 + 0.1 * 3 + 0.1 * 4 + 0.06 * 5 + 0.04 * 5 = 2.2 bits/symbol

2. Total no. of bits to be transmitted

   10 * 10 * 2.2 = 220 bits

3. Entropy = 2.1396

4. How much you saved = $\dfrac{10 * 10 * 5 - 10 * 10 * 2.2}{10 * 10 * 5}$ = 0.56 = 56%

INTERPIXEL REDUNDANCY

A picture of 5 randomly arranged pencils, and 5 parallel pencils have the same histogram, and same probabilities for each grey value.
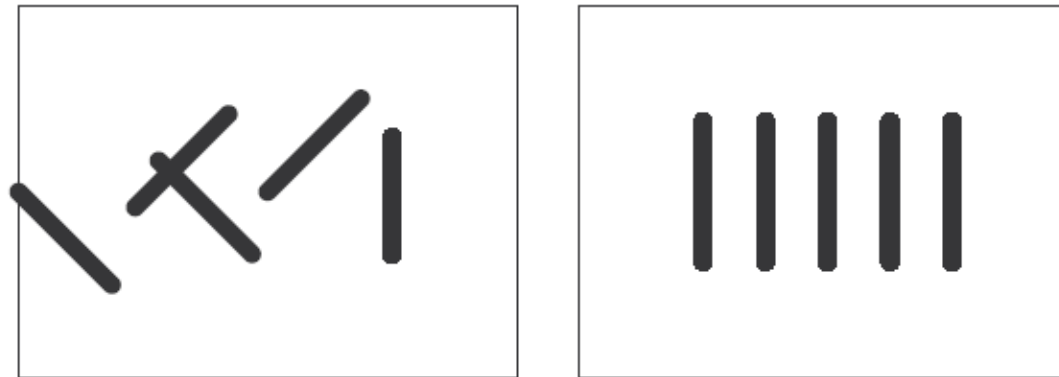
However in one case, the picture shows a strong spatial correlation. Hence value of next pixel can be predicted reasonably well. Then all values are not necessary, and many are redundant.

These inter-pixel redundancies can be removed by using not pictorial mapping like difference between adjacent pixels, value and length of grey level runs, etc. Usually this leads to a non-pictorial representation.

e.g 1024 bits of binary data can be coded as

(1,63)(0,87)(1,37)(0,5)(1,4)(0,556)(1,62)(0.210) requiring 88 bits.

RUN LENGTH ENCODING
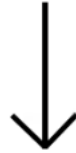
WHAT IS RUN-LENGTH ENCODING?

Run-length encoding (RLE) is a lossless compression method where sequences that display redundant data are stored as a single data value representing the repeated block and how many times it appears in the image.

Later, during decompression, the image can be reconstructed exactly from this information.

This type of compression works best with simple images and animations that have a lot of redundant pixels. It's useful for black and white images in particular.

For complex images and animations, if there aren't many redundant sections, RLE can make the file size bigger rather than smaller. Thus it's important to understand the content and whether this algorithm will help or hinder.

PSYCHO VISUAL REDUNDANCY

The eye and the brain do not respond to all visual information with same sensitivity.

Some information is neglected during the processing by the brain. Elimination of this information does not affect the interpretation of the image by the brain.

Edges and textural regions are interpreted as important features and the brain groups and correlates such grouping to produce its perception of an object.

Psycho visual redundancy is distinctly vision related, and its elimination does result in loss of information. Quantization is an example.

When 256 levels are reduced by grouping to 16 levels, objects are still recognizable. The compression is 2:1, but an objectionable graininess and contouring effect results.

IGS QUANTIZATION

The graininess from quantization can be reduced by Improved Grey Scale quantization which breaks up the edges and contours and makes them less apparent. IGS adds a pseudo random number generated from low order bits of neighboring pixels to each pixel before quantizing them. If most significant bits are all 1, the 0s are added. The msbs become the coded value.

Pixel	Gray Level	Sum	IGS Code
$i - 1$	N/A	0000 0000	N/A
i	0110 1100	0110 1100	0110
$i + 1$	1000 1011	1001 0111	1001
$i + 2$	1000 0111	1000 1110	1000
$i + 3$	1111 0100	1111 0100	1111

FIDELITY CRITERIA

The removal of psychovisually redundant data results in a loss of real or quantitative visual information. Because information of interest may be lost, a repeatable or reproducible means of quantifying the nature and extent of information loss is highly desirable. Two general classes of criteria are used as the basis for such an assessment:

A) Objective fidelity criteria

B) Subjective fidelity criteria.

OBJECTIVE FIDELITY CRITERION

When the level of information loss can be expressed as a function of the original or input image and the compressed and subsequently decompressed output image, it is said to be based on an objective fidelity criterion

Let the pixel error be given as $e(x, y) = \hat{f}(x, y) - f(x, y)$

Then total error between two images

$$\sum_{x=0}^{M-1}\sum_{y=0}^{N-1}\left[\hat{f}(x, y) - f(x, y)\right]$$

The RMS error is

$$e_{rms} = \left[\frac{1}{MN}\sum_{x=0}^{M-1}\sum_{y=0}^{N-1}\left[\hat{f}(x, y) - f(x, y)\right]^2\right]^{\frac{1}{2}}$$

The mean square SNR is

$$SNR_{ms} = \frac{\displaystyle\sum_{x=0}^{M-1}\sum_{y=0}^{N-1}\left[\hat{f}(x, y)\right]^2}{\displaystyle\sum_{x=0}^{M-1}\sum_{y=0}^{N-1}\left[\hat{f}(x, y) - f(x, y)\right]^2}$$

The rms value of the signal-to-noise ratio, denoted SNRrms, is obtained by taking the square root of Eq. above.

SUBJECTIVE FIDELITY CRITERION

Although objective fidelity criteria offer a simple and convenient mechanism for evaluating information loss, most decompressed images ultimately are viewed by humans. Consequently, measuring image quality by the subjective evaluations of a human observer often is more appropriate. This can be accomplished by showing a "typical" decompressed image to an appropriate cross section of viewers and averaging their evaluations. The evaluations may be made using an absolute rating scale or by means of side-by-side comparisons of $f(x, y)$ and $f^\wedge(x, y)$.

ARITHMETIC CODING

Main Idea of Arithmetic Coding

Sequence: $S_1\ S_2\ S_3\ S_4\ ....$ Mapped to... $\longrightarrow$

1

— 0.6457351....

0

Each possible sequence gets mapped to a unique number in $[0,1)$

- The mapping depends on the prob. of the symbols
- You don't need to *a priori* determine all possible mappings

how to obtain the segment SM for the message $babc$ produced by the source {'a':0.2,'b':0.5,'c':0.3}.
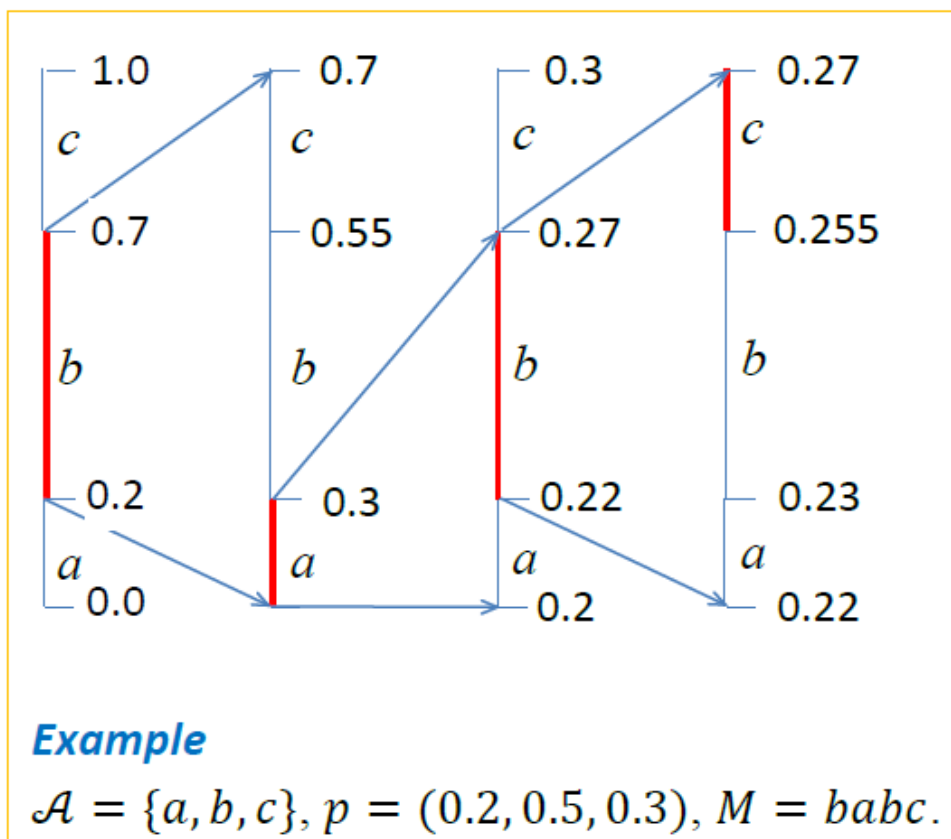
Let us proceed now to assign an interval to ba (second colum of the illustration), then to bab (third column) and finally to $babc$ (fourth column). The interval S_{ba} is obtained by subdividing

$$S_b = [l_b = 0.2, h_b = 0.7)$$

Example

$\mathcal{A} = \{a, b, c\}, p = (0.2, 0.5, 0.3), M = babc.$

ARITHMETIC CODING

into segments of relative length $p(a) = 0.2$, $p(b) = 0.5$ and $p(c) = 0.3$ and choosing the a-segment as S_{ba}. So the division points are

$l_{ba} = 0.2$, $h_{ba} = l_{bb} = 0.2 + 0.5 \times \sigma(a) = 0.30$,

$h_{bb} = l_{bc} = 0.2 + 0.5 \times \sigma(b) = 0.55$, $h_{bc} = 0.2 + 0.5 \times \sigma(c) = 0.70$, so

$S_{ba} = [0.2, 0.3)$, $S_{bb} = [0.30, 0.55)$, $S_{bc} = [0.55, 0.70)$.

Example

$\mathcal{A} = \{a, b, c\}$, $p = (0.2, 0.5, 0.3)$, $M = babc$.
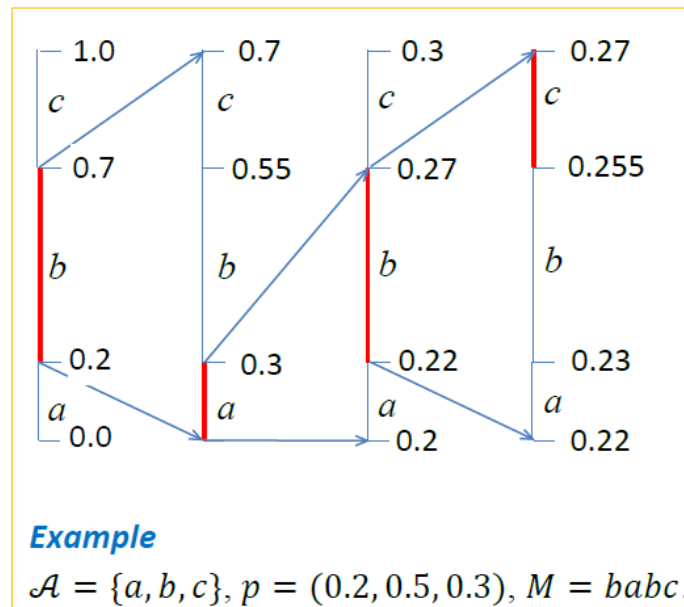
ARITHMETIC CODING

Now the interval S_{bab} is obtained in a similar way: subdivide S_{ba} into intervals that are proportional to $p(a)$, $p(b)$ and $p(c)$ and choose as S_{bab} the segment corresponding to b. Actually we have

$$h_{baa} = l_{bab} = 0.2 + 0.1 \times \sigma(a) = 0.22,$$
$$h_{bab} = l_{bac} = 0.2 + 0.1 \times \sigma(b) = 0.27,$$
$$h_{bac} = 0.2 + 0.1 \times \sigma(c) = 0.30,$$

and hence

$$S_{bab} = [l_{bab}, h_{bab}) = [0.22, 0.27).$$

Finally we get, following the same procedure with S_{bab},

$$S_{babc} = [0.22 + 0.05 \times \sigma(b), 0.22 + 0.05 \times \sigma(c)) = [0.255, 0.270).$$

Example
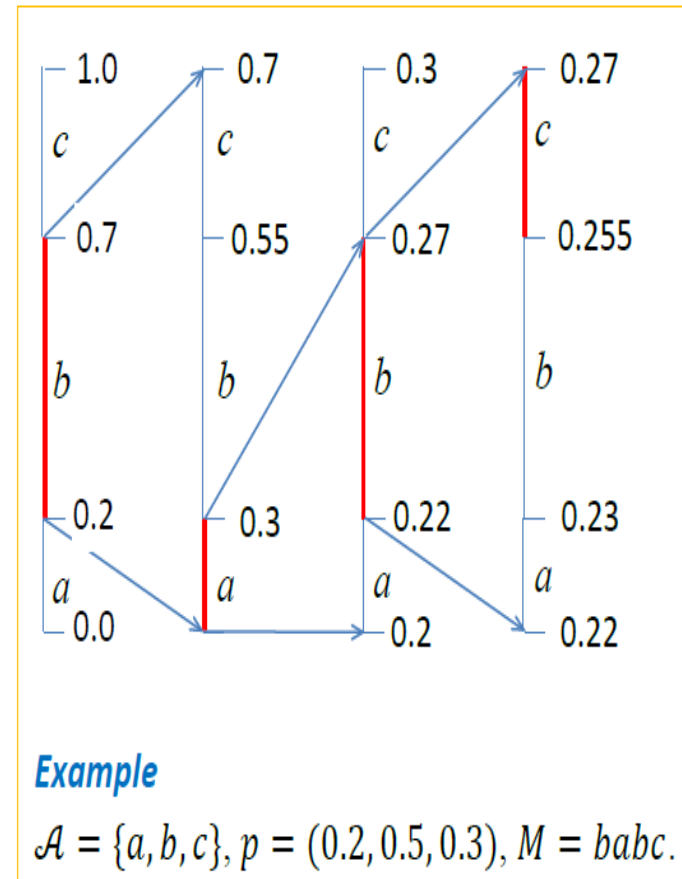
$\mathcal{A} = \{a, b, c\}, p = (0.2, 0.5, 0.3), M = babc.$

ARITHMETIC CODING
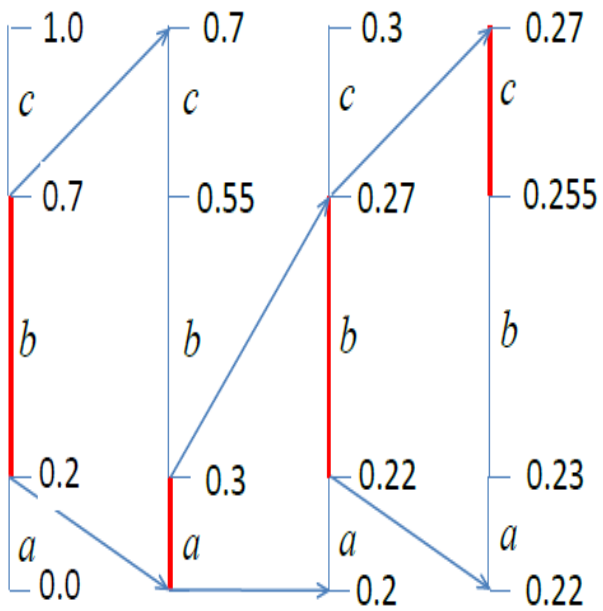
Example

$\mathcal{A} = \{a, b, c\}$, $p = (0.2, 0.5, 0.3)$, $M = babc$.

Interval	Subinterval			Symbol
[0,1]	[0.0-0.2]	[0.2-0.7]	[0.7-1.0]	
[0.2-0.7]	[0.2-0.3]	[0.3-0.55]	[0.55-0.7]	b
[0.2-0.3]	[0.2-0.22]	[0.22-0.27]	[0.27-0.3]	a
[0.22-0.27]	[0.22-0.23]	[0.23-0.255]	[0.255-0.27]	b
[0.255-0.27]				c

VECTOR QUANTIZATION

- *Vector quantization (VQ)* is the process of mapping a vector that can have many values to a vector that has a smaller (quantized) number of values

- For image compression, the vector corresponds to a small subimage, or block

VECTOR QUANTIZATION

EXAMPLE 10.3.3:

Given the following 4x4 subimage:

$$\begin{bmatrix} 65 & 70 & 71 & 75 \\ 71 & 70 & 71 & 81 \\ 81 & 80 & 81 & 82 \\ 90 & 90 & 91 & 92 \end{bmatrix}$$

This can be re-arranged into a 1-D vector by putting the rows adjacent as follows:

[row1   row2   row3   row4] = [65 70 71 75 71 70 71 81 81 80 81 82 90 90 91 92]

VECTOR QUANTIZATION

- VQ can be applied in both the spectral or spatial domains

- Information theory tells us that better compression can be achieved with vector quantization than with scalar quantization (rounding or truncating individual values)

VECTOR QUANTIZATION

- Vector quantization treats the entire subimage (vector) as a single entity and quantizes it by reducing the total number of bits required to represent the subimage

- This is done by utilizing a *codebook*, which stores a fixed set of vectors, and then coding the subimage by using the index (address) into the codebook

VECTOR QUANTIZATION

EXAMPLE 10.3.4:

Given an 8-bit, 256x256 image, we devise a vector quantization scheme that will encode each 4x4 block with one of the vectors in a codebook of 256 entries. We determine that we want to encode a specific subimage with vector number 122 in the codebook. For this subimage we then store the number 122 as the index into the codebook. Then when the image is decompressed, the vector at the 122 address in the codebook, is used for that particular subimage. This is illustrated in Figure 10.3-7. This will require 1 byte (8-bits) to be stored for each 4x4 block, providing a data reduction of 16 bytes for a 4x4 block to 1 byte, or 16:1

- In the example we achieved a 16:1 compression, but note that this assumes that the codebook is not stored with the compressed file

Figure 10.3-7: Quantizing with a Codebook
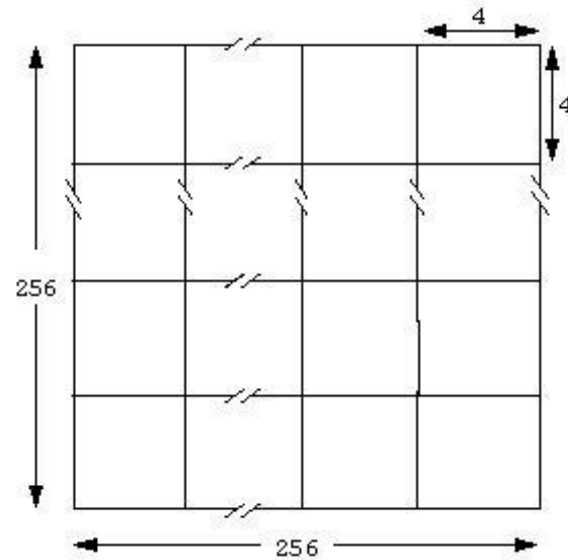
a) Original 256x256 image divided into 4x4 blocks
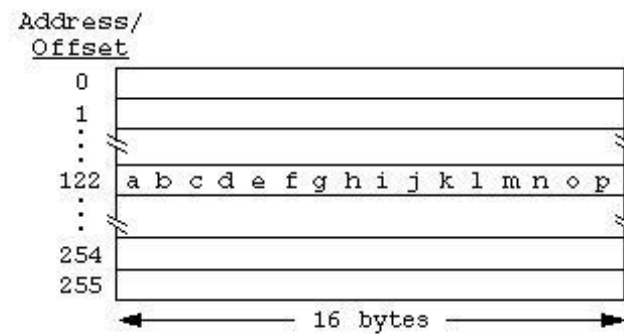
b) Codebook with 256 16-byte entries

c) A subimage decompressed with vector #122

VECTOR QUANTIZATION

- However, the codebook will need to be stored unless a generic codebook is devised which could be used for a particular type of image, in that case we need only store the name of that particular codebook file

- In the general case, better results will be obtained with a codebook that is designed for a particular image

EXAMPLE 10.3.5:

If we include the codebook in the compressed file from the previous example, the compression ratio will not be quite as good. For every 4x4 block we will have 1 byte. This gives us:

$$\left(\frac{256\ pixels}{4\ pixels/block}\right)\left(\frac{256\ pixels}{4\ pixels/block}\right) = 4096\ blocks$$

At 1 byte for each 4x4 block, this give us 4096 bytes for the codebook addresses. Now we also include the size of the codebook, 256x16:

$$4096 + (256)(16) = 8192\ bytes\ for\ the\ coded\ file$$

The original 8-bit, 256x256 image contained:

$$(256)(256) = 65,536\ bytes$$

Thus, we obtain a compression of:

$$\frac{65,536}{8192} = 8 \rightarrow 8 : 1\ compression$$

In this case, including the codebook cut the compression in half, from 16:1 to 8:1

VECTOR QUANTIZATION

- A training algorithm determines which vectors will be stored in the codebook by finding a set of vectors that best represent the blocks in the image

- This set of vectors is determined by optimizing some error criterion, where the *error* is defined as the sum of the vector distances between the original subimages and the resulting decompressed subimages

WHY JPEG IMAGE COMPRESSION?

The *compression ratio* of lossless methods (e.g., Huffman, Arithmetic, LZW) is not high enough for image and video compression.

JPEG uses *transform coding*, it is largely based on the following observations:

- Observation 1: A large majority of useful image contents change relatively slowly across images, i.e., it is unusual for intensity values to alter up and down several times in a small area, for example, within an 8 x 8 image block.
A translation of this fact into the spatial frequency domain, implies, generally, lower spatial frequency components contain **more information** than the high frequency components which often correspond to less useful details and noises.

- Observation 2: Experiments suggest that humans are more immune to loss of higher spatial frequency components than loss of lower frequency components.

JPEG CODING
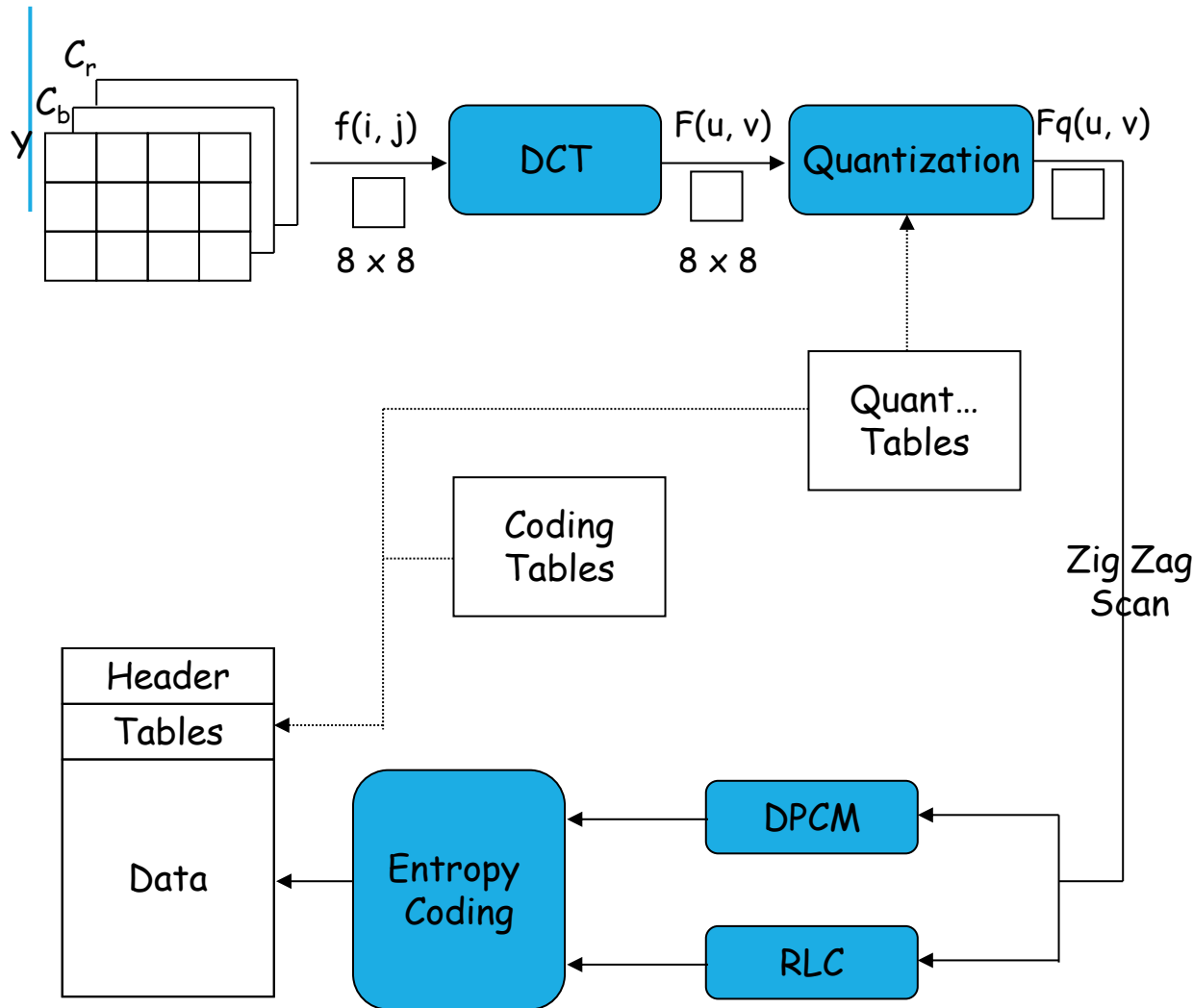
Steps Involved:

1. Discrete Cosine Transform of each 8x8 pixel array
   $f(x,y) \rightarrow_T F(u,v)$
2. Quantization using a table or using a constant
3. Zig-Zag scan to exploit redundancy
4. Differential Pulse Code Modulation(DPCM) on the DC component and Run length Coding of the AC components
5. Entropy coding (Huffman) of the final output

DCT : DISCRETE COSINE TRANSFORM

DCT converts the information contained in a block(8x8) of pixels from *spatial* domain to the *frequency* domain.

- A simple analogy: Consider a unsorted list of 12 numbers between 0 and 3 -> (2, 3, 1, 2, 2, 0, 1, 1, 0, 1, 0, 0). Consider a transformation of the list involving two steps (1.) sort the list (2.) Count the frequency of occurrence of each of the numbers ->(4,4,3,1). : Through this transformation we lost the spatial information but captured the frequency information.

- There are other transformations which retain the spatial information. E.g., Fourier transform, DCT etc. Therefore allowing us to move back and forth between spatial and frequency domains.

1-D DCT: 1-D Inverese DCT:

$$F(\omega) = \frac{a(u)}{2} \sum_{n=0}^{N-1} f(n)\cos\frac{(2n+1)\omega\pi}{16}$$

$$a(0) = \frac{1}{\sqrt{2}}$$

$$f'(n) = \frac{a(u)}{2} \sum_{\omega=0}^{N-1} F(\omega)\cos\frac{(2n+1)\omega\pi}{16}$$

$$a(p) = 1 \quad [p \neq 0]$$

EXAMPLE AND COMPARISON

DCT

FFT

100	-52	0	-5	0	-2	0	0.4

36	10	10	6	6	4	4	4

100	-52	0	-5	0	-2	0	0.4

36	10	10	6	6	4	4	4

Inverse DCT

Inverse FFT

8	15	24	32	40	48	57	63
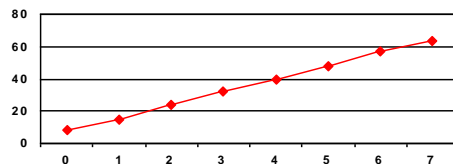
24	12	20	32	40	51	59	48

Example Description:
- r f(n) is given from n = 0 to 7; (N=8)
- r Using DCT(FFT) we compute F(ω) for ω = 0 to 7
- r We truncate and use Inverse Transform to compute f'(n)

2-D DCT

Images are two-dimensional; How do you perform 2-D DCT?
- Two series of 1-D transforms result in a 2-D transform as demonstrated in the figure below

f(i,j)
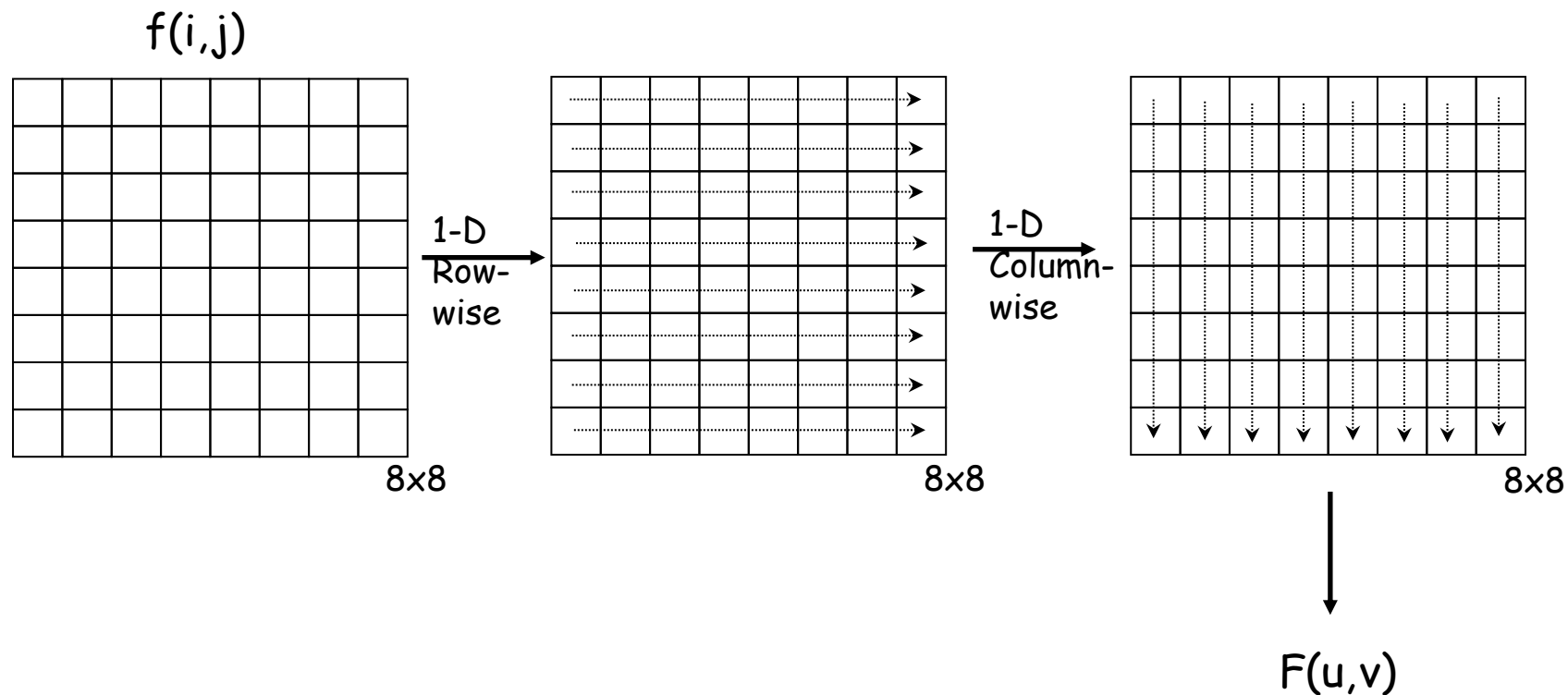
F(0,0) includes the lowest frequency in both directions is called DC coefficient Determines fundamental color of the block

F(0,1) …. F(7,7) are called AC coefficients Their frequency is non-zero in one or both directions

2-D TRANSFORM EXAMPLE

The following example will demonstrate the idea behind a 2-D transform by using our own cooked up transform: The transform computes a running cumulative sum.

f(i,j)

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	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
1	1	1	1	1	1	1	1

8x8

My Transform: $F_{my}(\omega) = \sum_{n=\omega}^{8} f(n)$

1-D Row-wise →

8	7	6	5	4	3	2	1
8	7	6	5	4	3	2	1
8	7	6	5	4	3	2	1
8	7	6	5	4	3	2	1
8	7	6	5	4	3	2	1
8	7	6	5	4	3	2	1
8	7	6	5	4	3	2	1
8	7	6	5	4	3	2	1

8x8

$\xrightarrow{\text{1-D Column-wise}}$

64	56	48	40	32	24	16	8
56	49	42	35	28	21	14	7
48	42	36	30	24	18	12	6
40	35	30	25	20	15	10	5
32	28	24	20	16	12	8	4
24	21	18	15	12	9	6	3
16	14	12	10	8	6	4	2
8	7	6	5	4	3	2	1

$F_{my}(u,v)$

8x8

Note that this is only a hypothetical transform. Do not confuse this with DCT

QUANTIZATION

Why? -- To reduce number of bits per sample

$F'(u,v) = round(F(u,v)/q(u,v))$

Example: 101101 = 45 (6 bits).
Truncate to 4 bits: 1011 = 11. (Compare 11 x 4 =44 against 45)
Truncate to 3 bits: 101 = 5. (Compare 8 x 5 =40 against 45)
Note, that the more bits we truncate the more precision we lose

Quantization error is the main source of the Lossy Compression.

Uniform Quantization:

- q(u,v) is a constant.

Non-uniform Quantization -- Quantization Tables

- Eye is most sensitive to low frequencies (upper left corner in frequency matrix), less sensitive to high frequencies (lower right corner)
- Custom quantization tables can be put in image/scan header.
- JPEG Standard defines two default quantization tables, one each for luminance and chrominance.

ZIG-ZAG SCAN

Why? -- to group low frequency coefficients in top of vector and high frequency coefficients at the bottom

Maps 8 x 8 matrix to a 1 x 64 vector

8x8

1x64
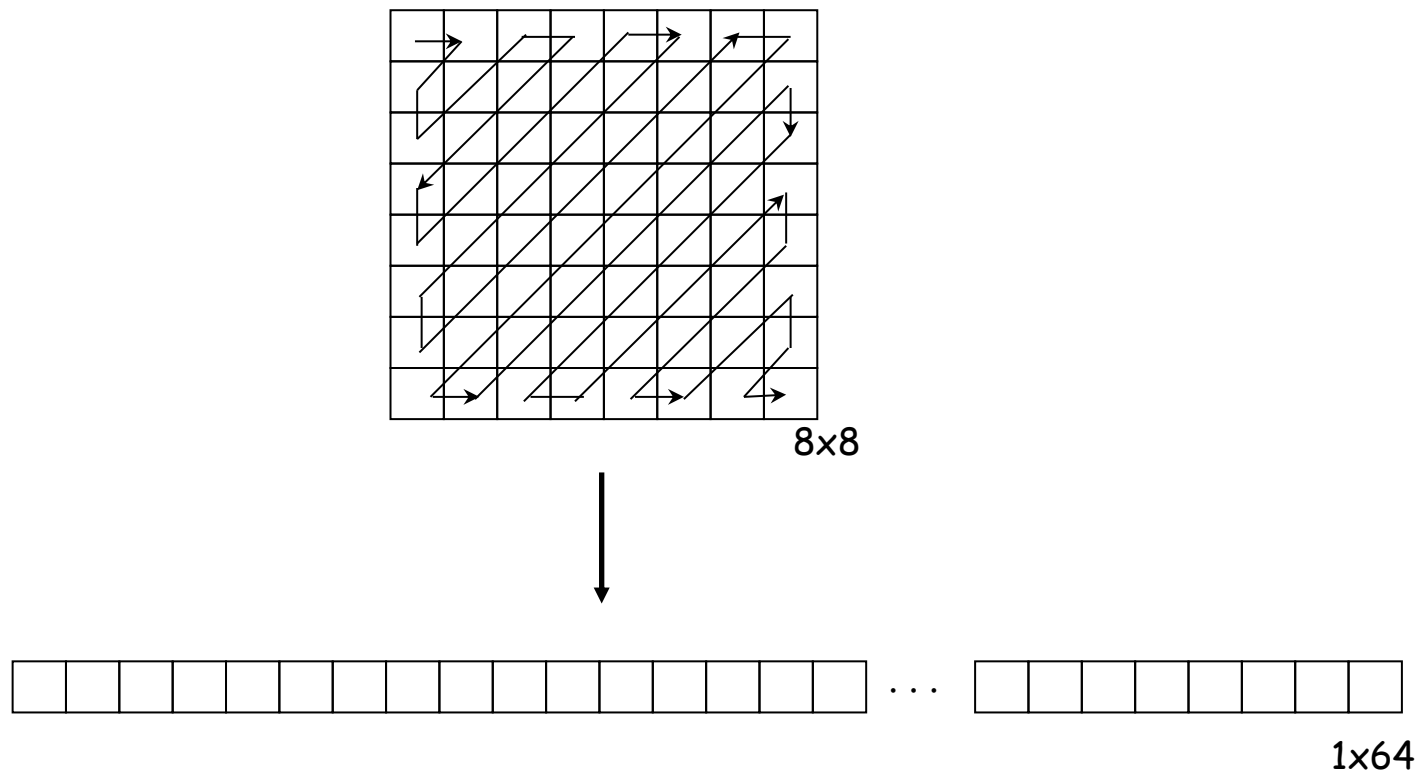
DPCM ON DC COMPONENTS

The DC component value in each 8x8 block is large and varies across blocks, but is often close to that in the previous block.
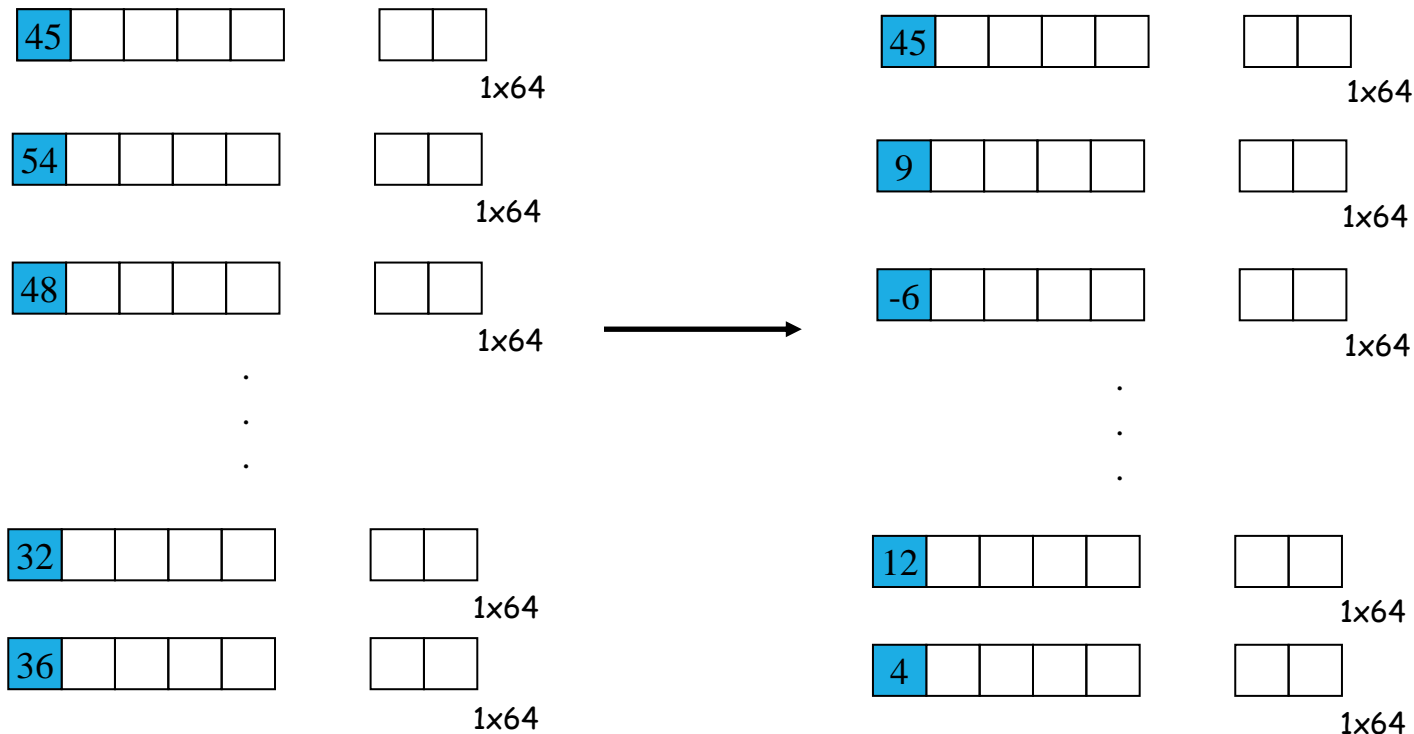
Differential Pulse Code Modulation (DPCM): Encode the difference between the current and previous 8x8 block. Remember, smaller number -> fewer bits

RLE ON AC COMPONENTS

The 1x64 vectors have a lot of zeros in them, more so towards the end of the vector.

- Higher up entries in the vector capture higher frequency (DCT) components which tend to be capture less of the content.
- Could have been as a result of using a quantization table

Encode a series of 0s as a (*skip,value*) pair, where *skip* is the number of zeros and *value* is the next non-zero component.

- Send (0,0) as end-of-block sentinel value.

1x64
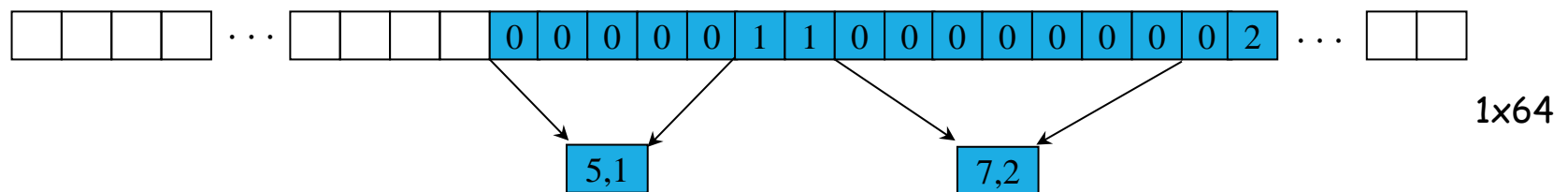
ENTROPY CODING: DC COMPONENTS

DC components are differentially coded as (**SIZE,Value**)

- The code for a **Value** is derived from the following table

SIZE	Value	Code
0	0	---
1	-1,1	0,1
2	-3, -2, 2,3	00,01,10,11
3	-7,…, -4, 4,…, 7	000,…, 011, 100,…111
4	-15,…, -8, 8,…, 15	0000,…, 0111, 1000,…, 1111
.		.
.		.
11	-2047,…, -1024, 1024,… 2047	…

Size_and_Value Table

ENTROPY CODING: DC COMPONENTS (CONTD..)

DC components are differentially coded as (**SIZE,Value**)

- The code for a **SIZE** is derived from the following table

SIZE	Code Length	Code
0	2	00
1	3	010
2	3	011
3	3	100
4	3	101
5	3	110
6	4	1110
7	5	11110
8	6	111110
9	7	1111110
10	8	11111110
11	9	111111110

Huffman Table for DC component SIZE field

Example: If a DC component is 40 and the previous DC component is 48. The difference is -8. Therefore it is coded as:

1010111

0111: The value for representing –8 (see Size_and_Value table)

101: The size from the same table reads 4. The corresponding code from the table at left is 101.

ENTROPY CODING: AC COMPONENTS

AC components (range −1023..1023) are coded as (S1,S2 pairs):

- **S1: (RunLength/SIZE)**
  - **RunLength:** The length of the consecutive zero values [0..15]
  - **SIZE:** The number of bits needed to code the *next* nonzero AC component's value. [0-A]
  - (0,0) is the End_Of_Block for the 8x8 block.
  - **S1** is Huffman coded (see AC code table below)
- **S2: (Value)**
  - **Value:** Is the value of the AC component.(refer to size_and_value table)

Run/ SIZE	Code Length	Code
0/0	4	1010
0/1	2	00
0/2	2	01
0/3	3	100
0/4	4	1011
0/5	5	11010
0/6	7	1111000
0/7	8	11111000
0/8	10	1111110110
0/9	16	1111111110000010
0/A	16	1111111110000011

Run/ SIZE	Code Length	Code
1/1	4	1100
1/2	5	11011
1/3	7	1111001
1/4	9	111110110
1/5	11	11111110110
1/6	16	1111111110000100
1/7	16	1111111110000101
1/8	16	1111111110000110
1/9	16	1111111110000111
1/A	16	1111111110001000
… 15/A	More	Such rows

ENTROPY CODING: EXAMPLE

40	12	0	0	0	0	0	0
10	-7	-4	0	0	0	0	0
1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Example: Consider encoding the AC components by arranging them in a zig-zag order -> 12,10, 1, -7 2 0s, -4, 56 zeros

12: read as zero 0s,12: (0/4)12 → 10111100

 1011: The code for (0/4 from AC code table)

 1100: The code for 12 from the Size_and_Value table.

10: (0/4)10 → 10111010

1: (0/1)1 → 001

-7: (0/3)-7 → 100000

2 0s, -4: (2/3)-4 → 1111110111011

 1111110111: The 10-bit code for 2/3

 011: representation of –4 from Size_and_Value table.

56 0s: (0,0) → 1010 (Rest of the components are zeros therefore we simply put the EOB to signify this fact)
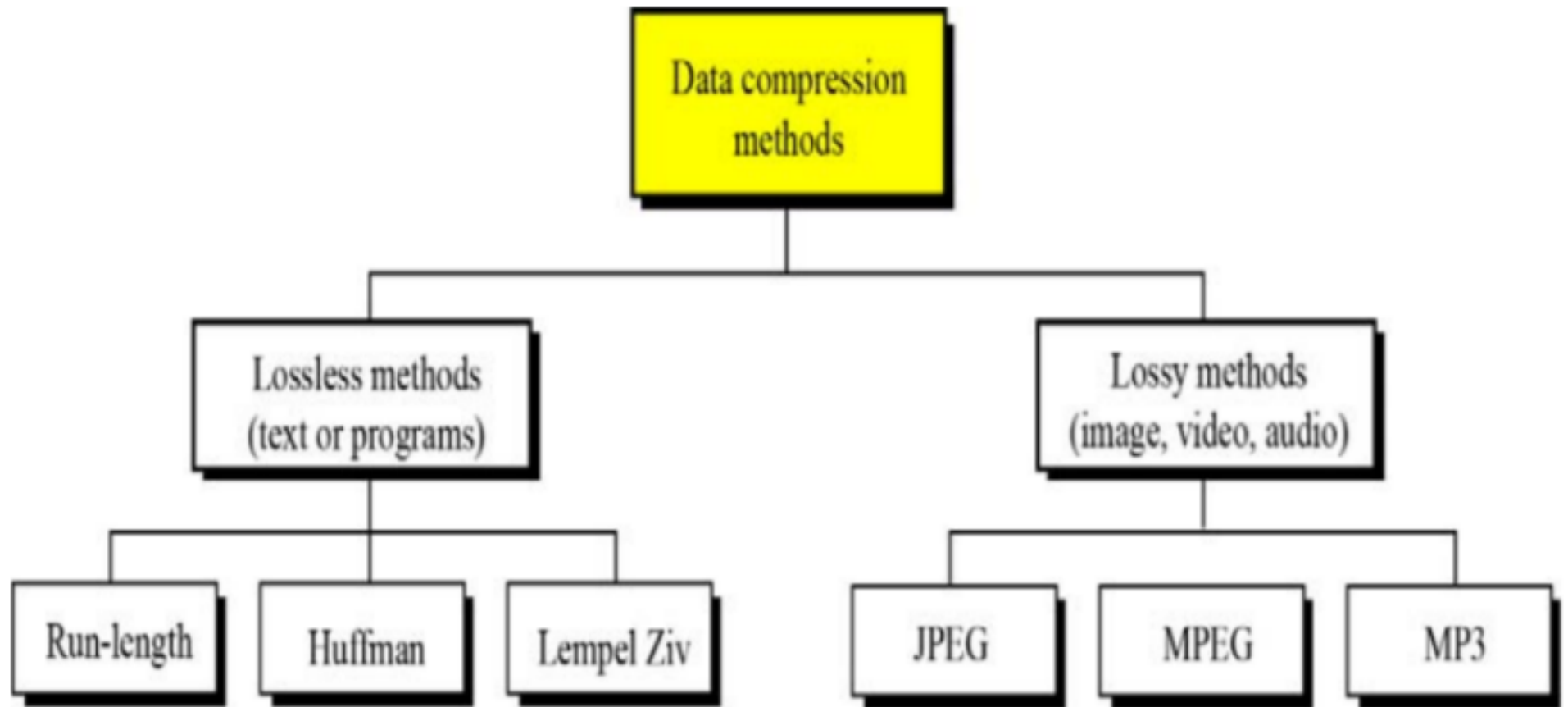
JPEG MODES

Sequential Mode:

Each image is encoded in a single left-to-right, top-to-bottom scan.

- The technique we have been discussing so far is an example of such a mode, also referred to as the **Baseline Sequential Mode.**

- It supports only 8-bit images as opposed to 12-bit images as described before.

IMAGE COMPRESSION METHODS

Lossless image compression techniques are:

1. Run-length encoding – used in PCX, BMP, TGA, TIFF
2. variable-length coding (Huffman, Arithmetic Coding)
3. Bit Plane coding
4. DPCM and Predictive Coding
5. Entropy encoding
6. LZW coding
7. Adaptive dictionary algorithms – used in GIF and TIFF
8. Deflation – used in PNG, MNG, and TIFF
9. Chain codes

Lossy compression techniques are:

1. Reducing the color space.
2. Chroma subsampling.
3. Fractal compression
4. Transform coding (DCT and Wavelet):