Unit 5: Data Compression

1. Need for Data Compression
2. Compression Basics
3. Storage Space
4. Coding Requirements
5. Lossless and Lossy Compression techniques
6. Source, Entropy and Hybrid Coding
7. Lossy Sequential OCT- based Mode
8. Expanded Lossy OCT-based Mode
9. JPEG and MPEG Compression

#Past Questions

- **2023 Q6**: Differentiate between lossless and lossy compression.
- **2024 Q6**: Compare arithmetic coding and Huffman coding.
- **2024 Q9**: Explain the JPEG compression process steps in detail with examples. [2+8]
- **2023 Q10**: Why is compression necessary for multimedia data? Explain steps involved in JPEG compression. [2+8]

1. Need for Data Compression

Data compression is the process of reducing the size of digital files to save **storage space** and **transmission time**.

Uncompressed data (graphics, audio, and video) require high storage capacity, which is not possible in today's technology like CD and DVD etc. Uncompressed data is also impossible for multimedia communication. These data require very high bandwidth over a digital network.

Data compression is essential in computing for efficient data storage and transmission. As data sizes increase, storage and bandwidth requirements grow. Compression helps reduce the volume of data to:

- **Save Storage Space**: Compressed files take up less disk or memory space.
  - *Example*: A 5 MB image can be compressed to 1 MB using JPEG.
- **Faster Transmission**: Smaller files transmit faster over networks.
  - *Example*: Compressed video files stream faster with less buffering.
- **Reduce Costs**: Lower storage and bandwidth needs reduce infrastructure costs.
- **Improve Performance**: Compression helps optimize backup, data archiving, and retrieval.

The most important compressed techniques in use today are JPEG for single picture, H. 263 For video, MPEG for video and audio etc.

---

2. Compression Basics

Compression techniques involve:

- **Encoder**: Converts original data into a compressed version.

- **Decoder**: Reconstructs the original data (exact or approximate) from the compressed form.

Compression Ratio = (Original Size) / (Compressed Size)

- *Example*: If a file of 100 KB is compressed to 25 KB, the ratio is 4:1.

Redundancy: Repeating patterns in data. Removing redundancy reduces size.
Irrelevancy: Removing data not perceived by humans (used in lossy compression).

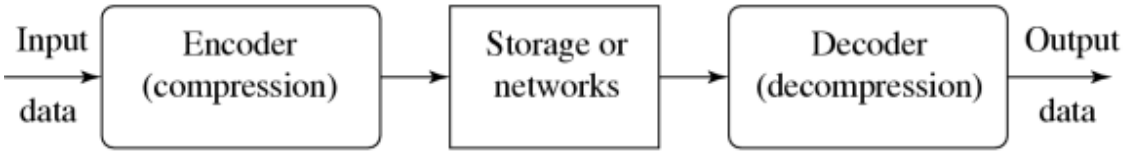


Fig 3.1 A general data compression scheme

---

3. Storage Space

Storage space refers to the **amount of memory or disk space** required to store data — whether it's the original (uncompressed) form or the compressed form.

- **In Data Compression:**
  The main goal of compression is to **reduce storage space** by encoding data more efficiently.
  - **Uncompressed data:** Takes more bytes (larger file size).
  - **Compressed data:** Takes fewer bytes (smaller file size).
- **Example:**
  - A raw image file might take **10 MB**.
  - After compression (e.g., JPEG), it might only need **2 MB**.
    → Thus, compression saves **8 MB of storage space**.
- **Key Point:**
  Efficient compression algorithms aim to **minimize storage space** while maintaining acceptable quality and enabling fast access.

---

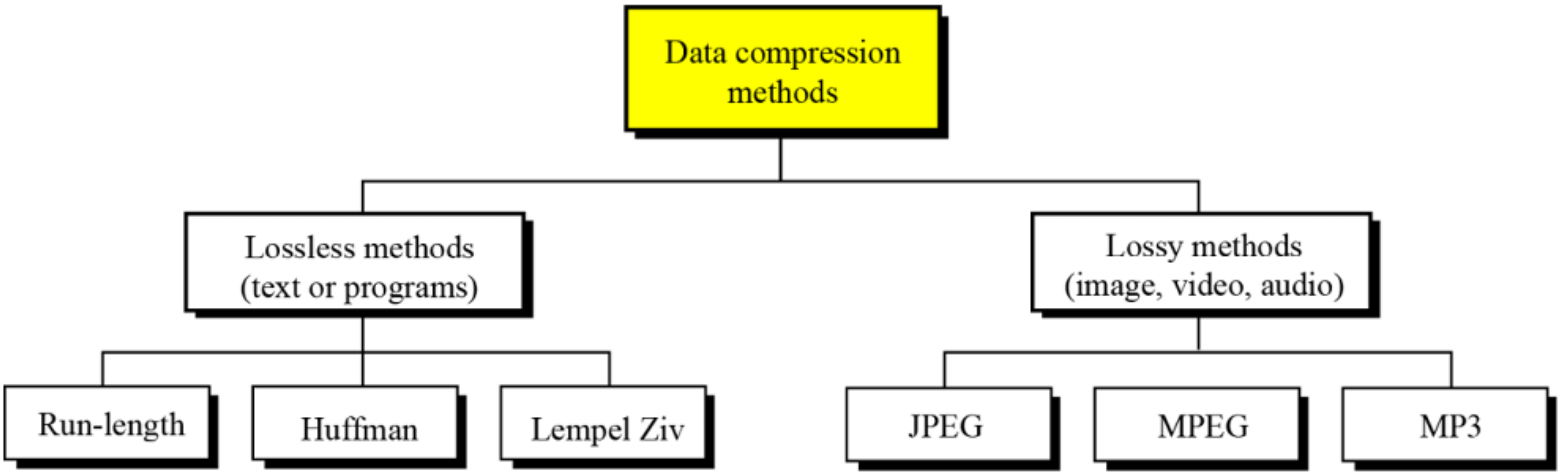
4. Coding Requirements

Coding requirements refer to the **rules, methods, and computational resources** needed to **encode (compress)** and **decode (decompress)** data.

- **In Data Compression:**
  Every compression technique has specific coding requirements, such as:
  - The **type of code** used (e.g., Huffman, Arithmetic, Run-Length Encoding).
  - The **complexity of the algorithm** (how much CPU time or memory it needs).
  - The **hardware or software support** needed to perform encoding/decoding.
- **Example:**
  - **Huffman Coding** requires building a frequency table and a binary tree — simple and fast.

- **Arithmetic Coding** gives higher compression but needs more computation power.

- **Key Point:**
  There's a **trade-off** between compression efficiency and coding requirements — better compression often needs **more complex coding**.



---

5. Lossless and Lossy Compression Techniques

Depending on whether the **original data can be perfectly recovered** or not after decompression, compression techniques are classified into two major types:

- **Lossless Compression**
- **Lossy Compression**

Lossless Compression

Lossless compression is a method in which the **original data can be completely reconstructed** from the compressed data **without any loss of information**.

In other words, after decompression, the data is **identical to the original**.

How It Works:

Lossless compression removes **redundancy** in data.
For example, repeated patterns, symbols, or bytes are replaced with shorter representations.

Example:

If we have a text: *AAABBBBCCCCCCDD*

→ It can be encoded as: *3A4B6C2D*

This is known as **Run-Length Encoding (RLE)**.

When decompressed, it gives back the **exact same data**.

Common Techniques:

1. **Run-Length Encoding (RLE)** – replaces consecutive repeated symbols with a count.
2. **Huffman Coding** – assigns shorter codes to frequent symbols and longer codes to rare ones.
3. **Lempel-Ziv-Welch (LZW)** – used in ZIP, GIF formats; replaces repeated patterns with dictionary codes.
4. **Arithmetic Coding** – represents entire messages as a single number between 0 and 1.

Used In: Text files (ZIP, GZIP), Executable files, Databases, Images (PNG, GIF)

Advantages:

- Perfect reconstruction (no quality loss)
- Suitable for critical data like text or software

Disadvantages:

- Compression ratio is lower compared to lossy methods
- Files remain relatively larger

Lossy Compression

Definition:

Lossy compression is a method where some **amount of data is lost permanently** during compression. The reconstructed data is **not identical**, but is **perceptually similar** to the original.

How It Works:

Lossy methods remove parts of the data that are **less important** or **less perceptible** to the human eye or ear.

For example:

- In images: small color variations invisible to the eye are removed.
- In audio: frequencies beyond human hearing are discarded.

Common Techniques:

1. **Transform Coding (DCT, Wavelet)** – converts signals into frequency components (used in JPEG, MPEG).
2. **Quantization** – reduces the precision of less important data.
3. **Predictive Coding** – encodes only the difference between predicted and actual values.

Used In: Images (JPEG), Audio (MP3, AAC), Video (MPEG, MP4)

Advantages:

- Very high compression ratio (much smaller files)
- Faster transmission over networks

Disadvantages:

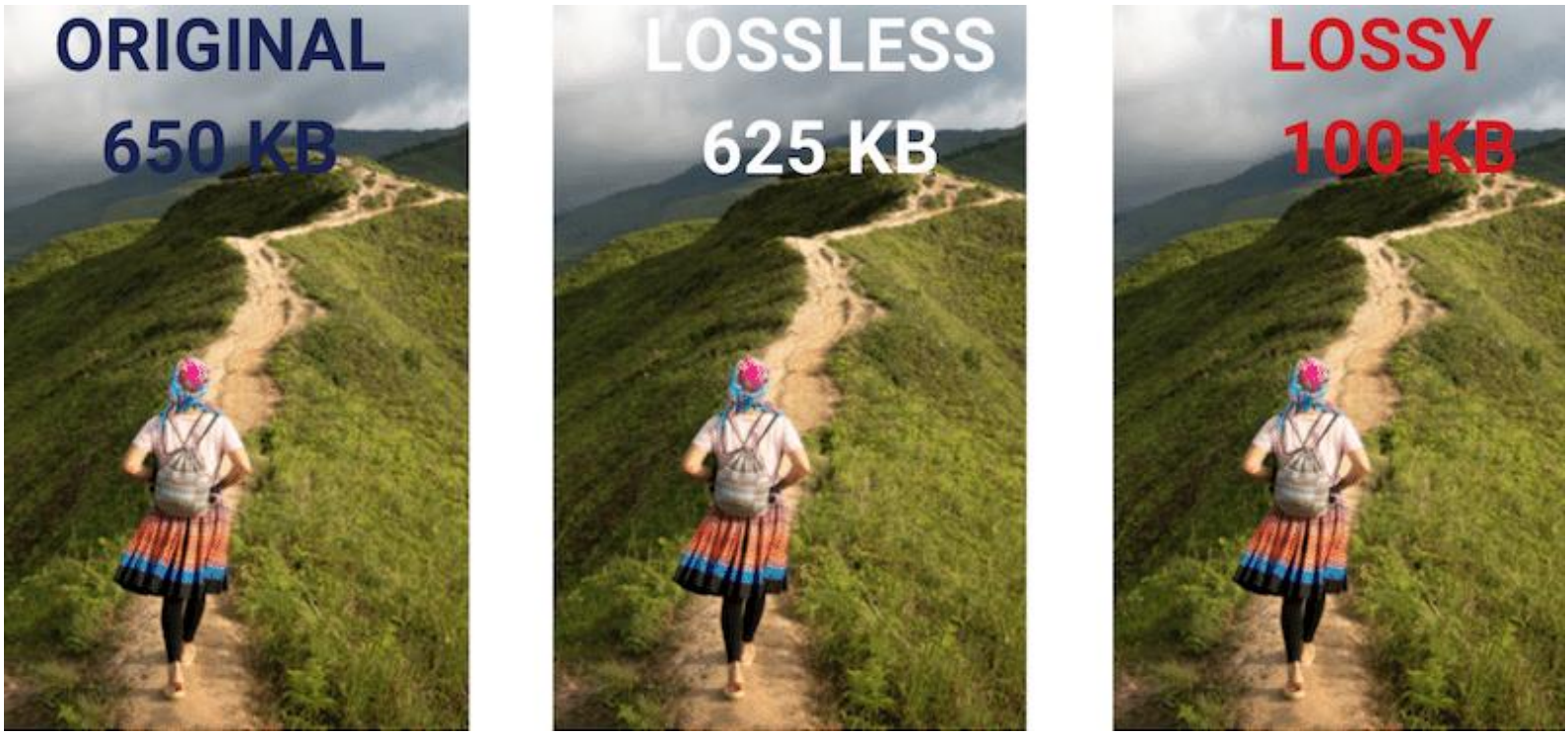
- Some loss of quality
- Not suitable for text or software
- Repeated compression → quality degradation

Key Differences Between Lossless and Lossy Compression

Feature	**Lossless Compression**	**Lossy Compression**
Data Recovery	100% exact restoration	Partial restoration
Data Loss	No information lost	Some information lost
Quality	Original quality retained	Slight to moderate quality reduction
Compression Ratio	Low to moderate	High
Used For	Text, program files, PNG, GIF	Audio, video, JPEG, MP3, MPEG
Example Techniques	RLE, Huffman, LZW	JPEG, MPEG, MP3
Suitability	When accuracy is essential	When small size and speed are important

Diagram (Conceptual Description):

Original Data ⟶ *Compression* ⟶ *Compressed Data* ⟶ *Decompression* ⟶ *Output*
Lossless: Output = Original (no loss)
Lossy: Output ≈ Original (some data lost)



Conclusion:

- **Lossless compression** is used when **data accuracy** is critical.
- **Lossy compression** is preferred when **space saving and faster transmission** are more important than exact accuracy (like in multimedia).

6. Source, Entropy, and Hybrid Coding

1. Source Coding (Lossy)

Definition:

Source coding (lossy) is a **compression technique** that reduces data size by **removing redundancies** or **less important information** based on the characteristics of the media.
It focuses on **retaining perceptually important information** while discarding data that has minimal effect on perceived quality.

Key Features:

- Some portion of the original data is **permanently removed** (cannot be recovered after decompression).
- Utilizes knowledge of **human perception**, such as:
  - The human eye is **less sensitive to color details** than brightness.
  - The human ear is **less sensitive to very high or low frequencies**.
- Focuses on the **semantics and structure** of the source (e.g., image, audio, or video).
- Provides a **high compression ratio** but results in **some quality degradation**.

Common Types of Lossy Source Coding:

Technique	Description
DPCM (Differential Pulse Code Modulation)	Encodes only the **difference between consecutive samples**, rather than the absolute values. This reduces redundancy in signals that change slowly.
DM (Delta Modulation)	A simplified form of DPCM that encodes the **difference as a single step (up or down)**, representing gradual signal changes.
DCT (Discrete Cosine Transform)	Converts the signal **to the frequency domain** and removes **high-frequency** (less important) parts. Commonly used in **JPEG image** and **MPEG video compression**.

Example:

In **JPEG compression**, DCT is applied to 8×8 image blocks.
High-frequency coefficients (which represent small color or brightness variations) are **discarded**, while low-frequency coefficients (representing main image features) are **kept** — resulting in smaller file sizes with minimal visual loss.

2. Entropy Coding (Lossless)

Entropy coding is a lossless compression technique that compresses data without losing any

information. It uses statistical properties of the data.

Key Features:

- No information is lost.
- Works on symbol frequencies.
- Regenerates the exact original data after decompression.
- Used in the final stages of compression pipelines.

Common Types:

a. **Run-Length Encoding (RLE)**
b. **Huffman Coding**
c. **Arithmetic Coding**

a. Run-Length Encoding (RLE)

- RLE is the **simplest form of entropy coding** used in **lossless data compression**.
- It compresses data by replacing **repeated bytes** with a **shorter representation**.
- Usually, 4–25 bytes of repetition can be reduced into just **3 bytes**.

Steps of RLE Compression

1. Scan the input data from left to right.
2. Count consecutive repeating characters .
3. If count ≥ 4, replace the run with: ***x!n ,*** Where
    - x → Repeating character
    - ! → Exclamation (flag)
    - n → Number of repetitions
      (**e.g**., A!5 for five A's)
4. If count < 4, write characters as they are (uncompressed).
5. Continue until the end of the input.

Example:

Original data: ABCCCCCDDDF → **length=11**

- A → 1 → Keep as A
- B → 1 → Keep as B
- C → 5 → Compress as C!5
- D → 3 → Keep as DDD
- F → 1 → Keep as F

Compressed Output: ABC!5DDDF → **length = 9**

Example 2:

Original Data: ABCCCCCDDDDDDF → **Length = 12**

After RLE Compression: ABC!5D!4F →**Length = 9**

So, the data is **compressed**.

Drawback of RLE

When there is **no repeating value**, then RLE generates **more of the output value (bigger size)** than the input value.

i.e., Compression of a data will **increase the size** compared to the original data.

Example: Original: `AABBCCD` **RLE:** `A02B01C02D01`

Original size: 7 bytes, Compressed size: 12 bytes → Compression failed

---

b. Huffman Coding

Huffman coding is a lossless data compression algorithm that assigns variable-length binary codes to characters based on their frequency of occurrence.

- Characters with higher frequency are assigned shorter codes.
- Characters with lower frequency are assigned longer codes.
- It uses prefix codes, meaning no code is a prefix of another. This ensures uniqueness and no ambiguity in decoding.

Purpose:

- To minimize the total number of bits used for representation.
- To ensure exact and complete reconstruction of the original data.
- Used in various compression formats like JPEG, MP3, and ZIP.

There are mainly two major parts in Huffman Coding

1. Build a Huffman Tree from input characters.
2. Traverse the Huffman Tree and assign codes to characters.

Huffman Coding Algorithm

1. **Input:** Characters and their frequencies.
2. **Create leaf nodes** for each character.
3. **Insert all nodes** into a min-priority queue (sorted by frequency).
4. **Repeat until only one node remains:**
    - Remove **two smallest** frequency nodes.
    - Create a **new node** with frequency = sum of the two.
    - Set the two nodes as **left** and **right** children.
    - Insert the new node back into the queue.
5. **The remaining node** is the root of the Huffman Tree.
6. **Assign codes:**
    - Traverse the tree:
        - **Left → 0**, **Right → 1**.
    - Record binary codes for each character.

Example:

Character	Frequency
a	05
b	09
c	12
d	13
e	16
f	45

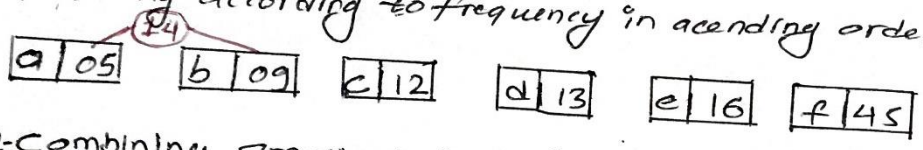
Here,
no. of character = 6
- $2^2 < 6$
- $2^3 > 6$

∴ There should be 3 bits for each character

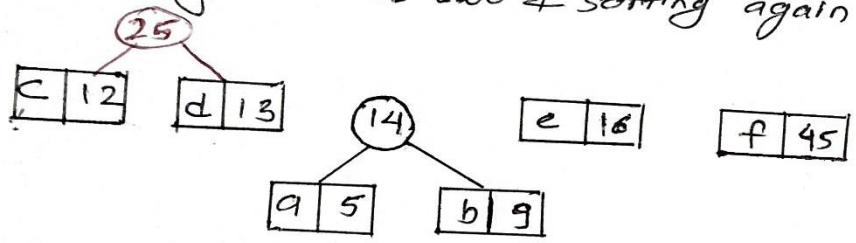
Thus, Total bits used = $5\times3 + 9\times3 + 12\times3 + 13\times3 + 16\times3 + 45\times3$
$= 300$ //

Soln:

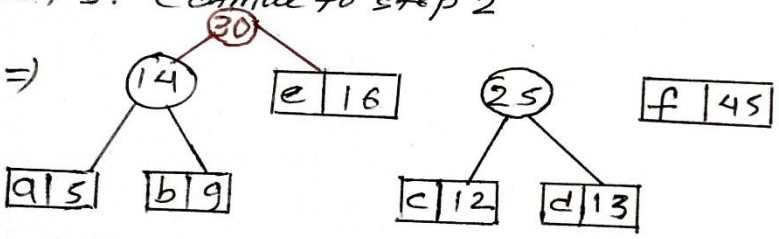
Step 1:- Sorting according to frequency in acending order



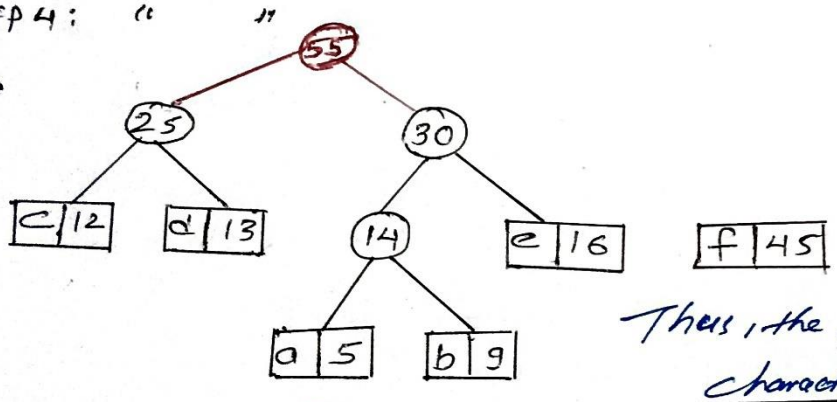
Step 2- Combining smallest two & sorting again



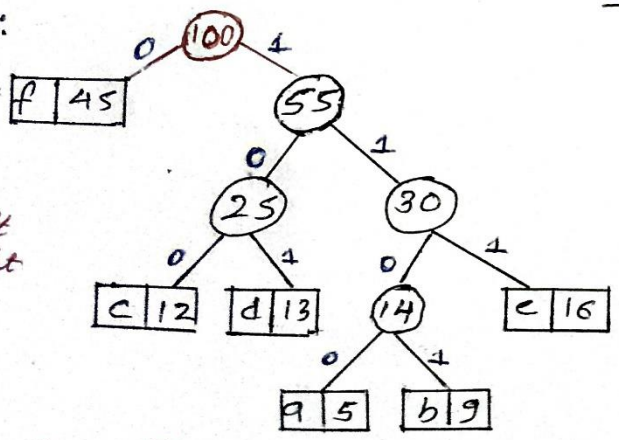
Step 3: Continue to step 2



Step 4: " "



Step 5:



Final tree

Assign
0 → left
1 → right

Thus, the codes are as follows:

Character	code-word	
a	1100	4 bits
b	1101	
c	100	3 bits
d	101	
e	111	
f	0	1 bit

∴ Total bits used = $5\times4 + 9\times4 + 12\times3 + 13\times3 + 16\times3 + 45\times0$
$= 179$ //

∴ $\frac{300-179}{300} \times 100 = 40.3\%$ of space is saved or compressed after using Huffman coding.

c. Arithmetic Coding

Arithmetic coding is a **lossless data compression technique** that encodes data by representing a sequence of symbols as a fraction (a real number) between 0 and 1. Unlike Huffman coding, which assigns fixed-length codes to symbols, arithmetic coding encodes the entire message as a **single number**, making it more efficient for certain types of data.

Key Points:

- **Lossless:** The original data can be fully recovered during the decompression process.
- **Representation of symbols:** Each symbol (e.g., character) in the input message is represented by a fractional range on the number line between 0 and 1.
- **Compression efficiency:** Symbols that appear more frequently in the input string will have smaller fractional ranges, using fewer bits, while less frequent symbols will take up larger ranges.

Process:

1. Assign probabilities to each symbol based on their frequencies.
2. Divide the number line from 0 to 1 according to these probabilities.
3. For each symbol in the input string, narrow the range of the number line to represent the symbol.
4. After processing the entire message, the resulting number represents the entire string in a fractional format.

Example:

- **Input:** ABABA
- **Probabilities:** A (0.6), B (0.4)
- **First symbol:** A → range from 0 to 0.6.
- **Second symbol:** B → range from 0.6 to 1.
- Continue this process until the entire string is represented by a single number between 0 and 1.

Arithmetic Coding vs Huffman Coding

Points	Arithmetic Coding	Huffman Coding
Concept	Represents the whole message as a single fractional value between 0 and 1.	Represents each symbol with a unique variable-length binary code.
Bit Usage	Uses fractional bits, giving near-optimal compression.	Uses whole bits, so compression may be slightly less efficient.
Efficiency	More efficient for data with close symbol probabilities.	Works well when symbol probabilities differ significantly.
Complexity	Complex to implement and decode.	Simple and easy to implement using binary trees.
Output	Produces a single number (e.g., 0.46).	Produces a set of binary codes (e.g., A=0, B=10).

3. Hybrid Coding (Entropy + Source Coding)

Hybrid coding is a **data compression technique** that combines both **source coding** and **entropy coding** to achieve **better compression efficiency**.

Key Features:

- **Step 1:** Source coding removes **perceptually unimportant or redundant data** (lossy compression).
- **Step 2:** Entropy coding then **compresses the remaining data losslessly** (e.g., Huffman, Arithmetic coding).
- Used widely in **modern multimedia compression systems**.

Advantages:

- High compression ratio
- Good image and video quality
- Efficient transmission and storage

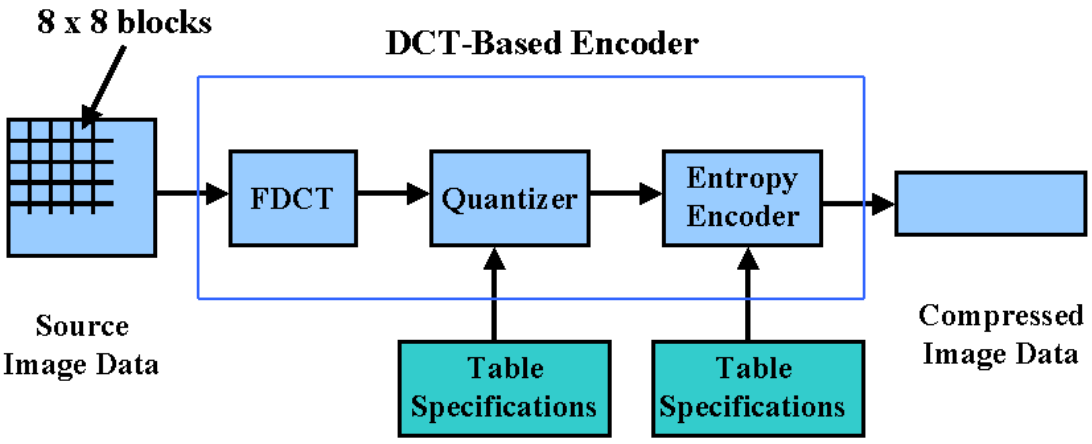
Common Standards (Examples):

Standard	Usage / Technique
JPEG	Image compression using **DCT + Huffman coding**
MPEG	Video compression using **DCT + motion estimation + entropy coding**
H.261	Used in **video conferencing**
DVI	Used in **digital video compression**

---

7. Lossy Sequential OCT-Based Mode

- **OCT = Orthogonal Transform Coding**, such as DCT (Discrete Cosine Transform).



DCT-Based Encoder Processing Steps

Lossy Sequential DCT-based mode is the **standard compression technique** used in **JPEG** for still images.
It works by converting image data into **frequency components** and **discarding less important details**, achieving high compression with minimal visual loss.


Original image


Image split into multiple 8x8 pixel blocks

Processing Steps:

1. Image Preparation and Blocking

- The image is divided into **8×8 pixel blocks**.
- Each pixel has an **8-bit intensity value (0–255)**.
- This block-wise division allows the frequency transformation (DCT) to be applied locally.

2. Forward Discrete Cosine Transform (FDCT)

- Each 8×8 block is transformed from the **spatial domain** (pixels) to the **frequency domain**.
- Produces 64 DCT coefficients:
  - **Low frequencies** → represent important visual information.
  - **High frequencies** → represent fine details and noise.

DCT Formula:

$$F(u,v) = \frac{1}{4}C(u)C(v)\sum_{x=0}^{7}\sum_{y=0}^{7} f(x,y)\cos\left[\frac{(2x+1)u\pi}{16}\right]\cos\left[\frac{(2y+1)v\pi}{16}\right]$$

where

$C(u), C(v) = \frac{1}{\sqrt{2}}$ when u or v = 0, else 1.

3. Quantization (Lossy Step)

- Each DCT coefficient is divided by a **quantization matrix value (Qtable)** and rounded.
- Formula:

$$Q(u,v) = \text{round}\left(\frac{T(u,v)}{Qtable(u,v)}\right)$$

Effects of Quantization:

- Reduces precision (especially of high frequencies).
- Preserves lower frequencies for visual clarity.
- Achieves large compression with minimal quality loss.
- Exploits the fact that the **human eye is less sensitive** to high-frequency details.

4. Entropy Encoding (Lossless Step)

After quantization, the data is further compressed **losslessly** using:

- **DC Coefficients:** Encoded as the **difference** from the previous block's DC value.
- **AC Coefficients:**
  - Scanned in **zig-zag order** to group zeros.
  - Encoded using **Run-Length Encoding (RLE)** followed by **Huffman Coding**.

Summary:

Step	Operation	Type	Purpose
1	Blocking	Pre-processing	Divide image into 8×8 blocks
2	FDCT	Transform	Convert pixels to frequency domain
3	Quantization	Lossy	Remove less important details
4	Entropy Encoding	Lossless	Compress data efficiently

---

8. Expanded Lossy OCT-Based Mode

- Image processing in this mode differs from the sequential DCT mode in terms of the number of bits per sample. Image preparation must use **p=12 bits** per pixel instead of the p=8 bits per pixel.
- **Arithmetic encoding** can be used in addition to Huffman coding in the entropy encoding section.

Note: Only the above two points are different; the rest are all the same as the sequential DCT-based mode.

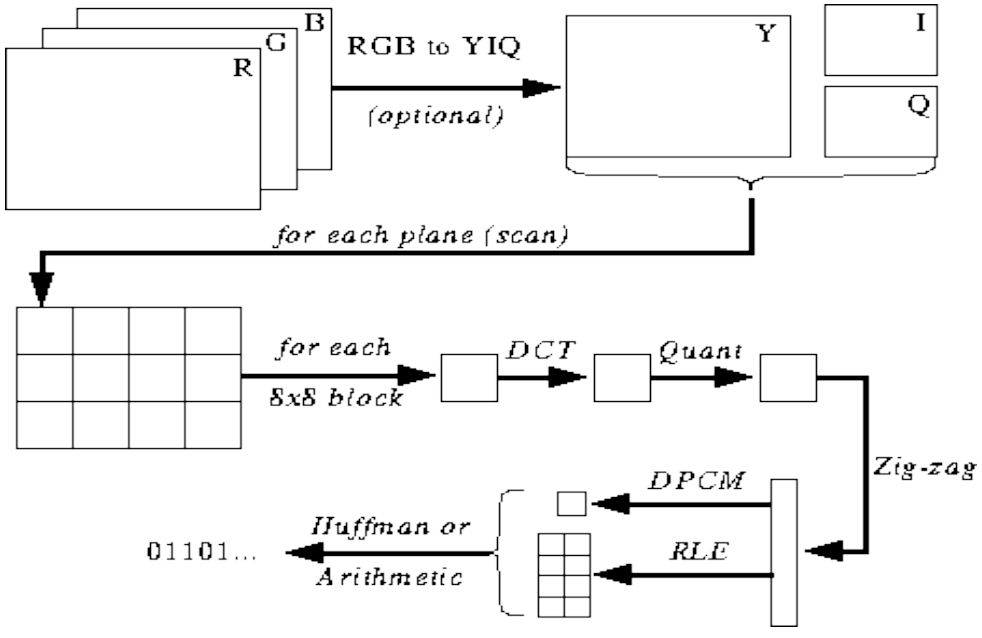
---

9. JPEG and MPEG Compression

JPEG (Joint Photographic Experts Group)

JPEG is a standard method of **compressing still images** (photographs).
It mainly uses **DCT (Discrete Cosine Transform)** to convert spatial domain data (pixels) into frequency domain data and then applies **quantization** and **entropy coding**.

Objective: To reduce the file size of photographic images with **minimal visible quality loss**.



Steps in JPEG Compression:

Step 1 — Split image into 8×8 blocks

- Divide the image into non-overlapping blocks of **8×8 = 64 pixels** (Minimum Coded Units, MCUs).
- Each pixel value is 0–255 (8 bits).
- Processing is done block-by-block.

Step 2 — Color space conversion (RGB → YCbCr)

- Convert RGB to **YCbCr** because JPEG uses a Y-Cb-Cr model instead of RGB.
  - **Y** = luminance (brightness)
  - **Cb** = blue difference (chrominance)
  - **Cr** = red difference (chrominance)
- **Reason:** Human eyes are more sensitive to brightness than color.

Step 3 — Discrete Cosine Transform (DCT)

- Apply **8×8 Forward DCT** to each block → transforms spatial pixel values to frequency coefficients.
- Result: 64 coefficients per block:
  - Top-left: **low frequencies** (most visually important).
  - Bottom-right: **high frequencies** (fine details / noise).

Step 4 — Quantization (lossy)

- Divide each DCT coefficient by the corresponding entry in the **Quantization Table** and round:

$$Q(u,v) = \text{round}\big(T(u,v)/Qtable(u,v) \big)$$

- Low frequencies use smaller divisors (preserved); high frequencies use larger divisors (often become zero).
- **This is the lossy step** — main source of perceptual quality loss and compression.

Step 5 — Zig-zag scanning

- Convert the 8×8 quantized matrix into a 1×64 vector using **zig-zag order** so low-frequency coefficients appear first and zeros cluster later.
- Example partial zig-zag index order (flat indices 0..63):

  0, 1, 5, 6, 14, 15, 27, 28,
  2, 4, 7, 13, 16, 26, 29, 42, ...

(You can draw the full 8×8 zig-zag map in notes; its purpose is grouping nonzeros at front.)

Step 6 — Differential coding (DPCM) for DC coefficient

- The **DC coefficient** (first element in vector) represents the block's average.
- Store **difference** between current block DC and previous block DC:

$$\text{DPCM} = \text{DC}_{\text{current}} - \text{DC}_{\text{previous}}$$

- Differences are usually small → compressible.

Step 7 — Run-Length Encoding (RLE) for AC coefficients

- After zig-zag, many trailing values are zero (from quantization).
- Use RLE to encode sequences of zeros as **(run, value)** pairs:
  - Example: [12, 0, 0, 0, 5, 0, 0, 3] → (0,12), (3,5), (2,3)
- Often followed by an **End-Of-Block (EOB)** marker when the rest are zeros.

Step 8 — Huffman encoding (entropy coding)

- Final compression step.
- DC and AC symbols are encoded using Huffman coding, which assigns shorter codes to more frequent values.

Advantages (brief)

- High compression ratio with good visual quality.
- Simple decoder, widely supported (web, cameras, phones).
- Flexible: different quality settings via quantization tables.

Notes / Limitations

- Blocking artifacts appear at high compression (visible 8×8 blocks).
- Not ideal for images with sharp edges or text — loss may be noticeable.
- Progressive JPEG allows multi-scan progressive display (low→high quality while loading).

MPEG (Moving Picture Experts Group)

MPEG is a **video compression technique** that compresses a **sequence of images (frames)** rather than just one.
It reduces both **spatial redundancy** (within a frame, like JPEG) and **temporal redundancy** (between frames).

Common MPEG Versions:

Version	Used In	Description
MPEG-1	Video CDs (VCDs)	Suitable for low-resolution video.
MPEG-2	DVDs, digital TV	Higher quality for broadcasting.
MPEG-4	Web, mobile, streaming (.mp4)	Efficient compression with better quality.

Steps of MPEG Compression

Step 1 — Resolution Reduction (Color Space Conversion)

- Converts frames from **RGB → YUV** color space.
- Components:
  - **Y** = luminance (brightness)

    - **U**, **V** = chrominance (color details)
- **Y** is preserved more accurately because the human eye is more sensitive to brightness than color.
- U and V components are **subsampled** to reduce data (e.g., 4:2:0 format).

Step 2 — Motion Estimation

- MPEG doesn't store every frame fully.
- It compares the **current frame** with the **previous frame** and finds **moved blocks** of pixels.
- Calculates **motion vectors** showing how each block has shifted.
- Removes **temporal redundancy** (repeated content across frames).

***Example**:* If an object moves slightly to the right, MPEG just stores "object moved 5 pixels right" instead of saving the whole new image.

Step 3 — Motion Compensation & Image Substitution

- Uses motion vectors to **reconstruct new frames** based on reference (previous) frames.
- Instead of storing the entire frame, stores only the difference from the reference frame.
- Reduces file size and bandwidth requirements.
- Greatly reduces the amount of data required to represent motion.

Step 4 — Discrete Cosine Transform (DCT)

- Similar to JPEG.
- Each frame (or residual block) undergoes **8×8 DCT** to convert spatial domain pixels into frequency domain data.
- Low-frequency components (important details) are preserved; high-frequency ones (fine details/noise) are reduced.

Step 5 — Quantization (Lossy Step)

- Each DCT coefficient is divided by a quantization matrix and rounded.
- Removes less noticeable (high-frequency) information to reduce data size.
- This is the **main lossy** part of MPEG compression.

Step 6 — Run-Length Encoding (RLE)

- Converts long runs of zeros in AC coefficients into **(skip, value)** pairs.
- Reduces data size by representing sequences compactly.

Step 7 — Entropy Encoding

- Finally, applies **Huffman coding** (or sometimes Arithmetic coding) for lossless compression.
- Shorter codes are assigned to frequent symbols, removing redundancy.

Types of MPEG Frames

To handle motion efficiently, MPEG uses 3 kinds of frames:

Frame Type	Full Form	Description
I-Frame	Intra-coded	Like a JPEG image; no reference to other frames.
P-Frame	Predictive-coded	Encoded using motion data from a previous I or P frame.
B-Frame	Bi-directional	Encoded using both previous and next frames for maximum compression.

Advantages of MPEG Compression

- Very **high compression ratio** (10×–50× smaller than raw video).
- **Smooth motion** with fewer frames stored.
- Efficient for **streaming, broadcasting, and storage**.
- Maintains **good visual quality** with less data.

Limitations

- **Lossy compression** — quality decreases after many re-encodings.
- **Complex encoding/decoding** due to motion estimation.
- Sensitive to **transmission errors** (as frames depend on others).

MPEG vs JPEG

Feature	JPEG	MPEG
Purpose	Compresses **still images**	Compresses **video and audio**
Compression Type	Uses **intra-frame compression** (within one image)	Uses **inter-frame compression** (between multiple frames)
Redundancy Removed	Removes **spatial redundancy**	Removes **spatial and temporal redundancy**
Techniques Used	DCT, Quantization, Zigzag, Run-Length Encoding (RLE), Huffman Coding	Motion Estimation & Compensation, DCT, Quantization, Entropy Coding
Output Formats	.jpg, .jpeg	.mpg, .mpeg, .mp4

Note:

Spatial Redundancy Within one frame — reduced using **DCT and quantization**.

Temporal Redundancy Between successive frames — reduced using **motion estimation and prediction**.