# Simplified Video Compression with DCT and Predictive Coding

Intro to Digital Signal Processing Project

May 5, 2025

## Contents

# 1 Introduction

**Important Note:** This project **does not have to be individual project**. You are allowed to group up with another student. Groups of up to 2 people are allowed. Only a single submission from each group is sufficient. Make sure to include both names and student numbers in your reports.

In this project, you are going to implement a simple video compression algorithm in MATLAB. Many of you probably heard of keywords such as MPEG-4 or H264 in the context of video files. These are names of lossy video compression standards. Algorithms such as these are what makes the current video streaming services possible as we can stream 4K 60 FPS videos with an internet speed of 50 megabits per second from our phones.

Some of the concepts are briefly introduced in Section 2. However, while implementing, this document might not be sufficient. For the general overview of the project, skip to Section 3.

# 2 Concepts & Definitions

## 2.1 Digital Video Structure

A video is an ordered sequence of images which we call frames. In this project we will work on a video 480x360 (width, height) video. The frames of the video is provided to you as **.jpg** images. MATLAB's Image Processing Toolbox will be helpful for this project.

MATLAB's `imread` function reads images in (H, W, c) order with data type uint8. This means that the frames you read will be in dimensions (360, 480, 3). The third dimension is the color dimension (in the order red, green, blue).

We will use macroblocks of size 8x8. A macroblock is simply a submatrix of the image. We will process the frames by dividing them into "non-overlapping macroblocks". This will be explained in more detail later.

## 2.2 Transform Coding

Transform coding is a core technique in image and video compression. It exploits spatial redundancy by transforming image blocks into a frequency domain representation, where perceptually less important components can be discarded or coarsely encoded.

### 2.2.1 Discrete Cosine Transform (DCT)

The Discrete Cosine Transform (DCT) is used to convert spatial-domain image data into the frequency domain. For an $8 \times 8$ image block $B$, the 2D DCT produces a new matrix $C$ of the same size. The coefficients $C_{u,v}$ represent the block's frequency content, where low frequencies are concentrated toward the top-left.

In practice, most of the visually important information is retained in a few low-frequency DCT coefficients. High-frequency coefficients can be approximated or discarded.

In MATLAB, `dct2` and `idct2` functions can be used to calculate DCT and its inverse transform.

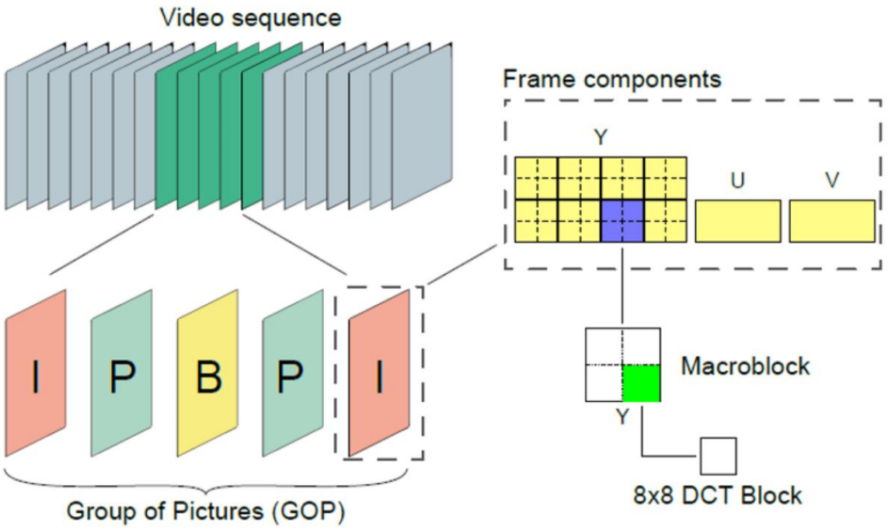


Figure 1: Figure describing general concepts. (Taken from https://blog.ampedsoftware.com/2023/07/19/how-to-use-the-macroblocks-filter-in-amped-five)

### 2.2.2 Quantization

Quantization is the process of reducing the precision of the DCT coefficients, introducing controlled loss of information to achieve compression. Each DCT coefficient is divided by a corresponding value from a quantization matrix (element-wise) and rounded to the nearest integer:

$$Q_{u,v} = \text{round}\left(\frac{C_{u,v}}{Q_{\text{mat}}(u,v)}\right)$$

Quantization reduces the number of bits needed to represent the DCT coefficients, especially at higher frequencies, where human vision is less sensitive to error. A coarser quantization matrix leads to greater compression at the expense of image quality.

$$\begin{bmatrix} 16 & 11 & 10 & 16 & 24 & 40 & 51 & 61 \\ 12 & 12 & 14 & 19 & 26 & 58 & 60 & 55 \\ 14 & 13 & 16 & 24 & 40 & 57 & 69 & 56 \\ 14 & 17 & 22 & 29 & 51 & 87 & 80 & 62 \\ 18 & 22 & 37 & 56 & 68 & 109 & 103 & 77 \\ 24 & 35 & 55 & 64 & 81 & 104 & 113 & 92 \\ 49 & 64 & 78 & 87 & 103 & 121 & 120 & 101 \\ 72 & 92 & 95 & 98 & 112 & 100 & 103 & 99 \end{bmatrix}$$

Figure 2: Quantization matrix for this project

In this project, we will use the quantization matrix (see Figure 2) provided in **this** WikiPedia article.

### 2.2.3 Zigzag Scan and Run-Length Encoding (RLE)

After quantization, the $8 \times 8$ block is converted into a 1D vector using a zigzag scan. This ordering places low-frequency coefficients (which are typically non-zero) at the beginning and high-frequency coefficients (usually zeros) at the end.

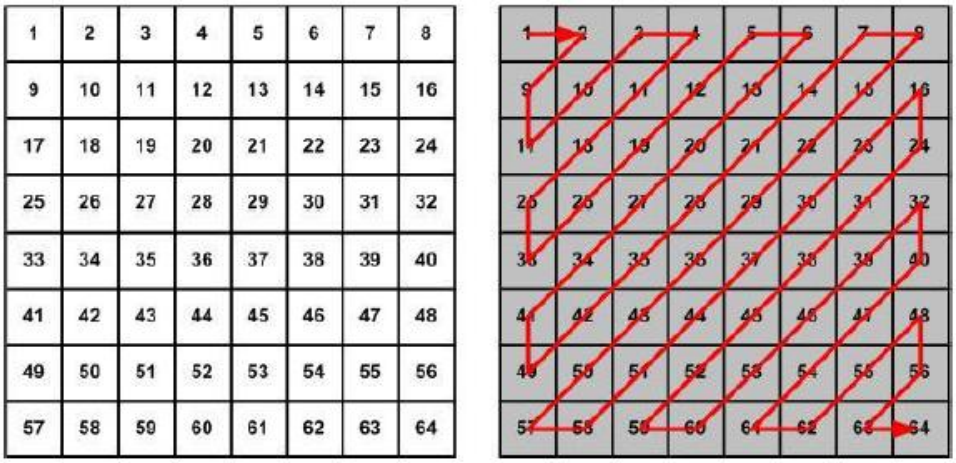


Figure 3: Zigzag ordering pattern for an $8 \times 8$ block

To further compress the data, Run-Length Encoding (RLE) is applied to the zigzag vector. RLE replaces consecutive zero values with a pair `(run_length, value)`, where `run_length` is the number of times `value` consecutively repeats in the original sequence. This significantly reduces the storage needed for sparse coefficient vectors.

**Example:**

```
Zigzag vector:  [12, -3, 0, 0, 0, 5, 0, 0, 0, 0, -1]
RLE output:     [(1,12), (1,-3), (3,0), (1,5), (4,0), (1,-1)]
```

This combination of DCT, quantization, zigzag scanning, and RLE forms the basis of many image and video compression standards, including JPEG and MPEG.

## 2.3 Intra- and Inter-frame Compression

Video compression exploits both spatial and temporal redundancy. Spatial redundancy within a frame is handled by transform coding techniques (e.g., DCT and quantization), while temporal redundancy between frames is addressed using inter-frame prediction.

### 2.3.1 I-Frame (Intra-coded Frame)

An I-frame is compressed independently of other frames. It serves as a reference and uses only intra-frame coding techniques. The image is divided into fixed-size macroblocks (e.g., $8 \times 8$), and each macroblock is processed as follows:

- Apply 2D Discrete Cosine Transform (DCT) to each macroblock.
- Quantize the DCT coefficients using a predefined quantization matrix.
- Reorder the quantized coefficients using zigzag scan.
- Apply Run-Length Encoding (RLE) to compress the sparse sequence.

I-frames provide random access points in a video stream and form the anchor for predictive coding of subsequent frames.

### 2.3.2 P-Frame (Predicted Frame)

P-frames exploit temporal redundancy by encoding only the *difference* (residual) between the current frame and a reference (usually previous) frame. This is done macroblock-by-macroblock without motion estimation in our simplified scheme. For each macroblock:

- Compute the residual by subtracting the corresponding macroblock from the previous frame:
$$R_{i,j} = B_{i,j}^{(t)} - B_{i,j}^{(t-1)}$$
- Apply DCT to the residual block.
- Quantize, zigzag, and RLE encode as in the I-frame.

During decoding, the residual is added back to the reconstructed previous macroblock to obtain the current macroblock:

$$B_{i,j}^{(t)} = \hat{B}_{i,j}^{(t-1)} + \hat{R}_{i,j}$$

This predictive approach reduces redundancy across time and improves compression efficiency, particularly for video sequences with minimal motion.

### 2.3.3 Group-of-Pictures (GOP)

A Group of Pictures (GOP) is a sequence of video frames consisting of different types of coded frames, typically beginning with an I-frame followed by one or more predictive (P) or bidirectional (B) frames. The GOP structure defines the temporal pattern of these frames and is a key factor in controlling compression efficiency and random access capability.

In our simplified scheme, we use only I-frames and P-frames. A typical GOP structure (GOP size = 5) might look like:

$$\text{GOP: } I_1\, P_1\, P_2\, P_3\, P_4$$

Each GOP begins with an I-frame, which is independently encoded, followed by several P-frames that store only residual data relative to the previous frame. This structure balances the trade-off between compression (favoring more P-frames) and random access (favoring more I-frames). The length of a GOP (i.e., number of frames between successive I-frames) affects:

- **Compression Ratio:** Longer GOPs generally yield higher compression by maximizing the use of temporal prediction.
- **Error Propagation:** Errors in one P-frame can affect all dependent future frames until the next I-frame.
- **Random Access:** Shorter GOPs allow for more frequent seeking and easier editing, since decoding can begin from any I-frame.

By organizing frames into GOPs, the encoder can efficiently manage temporal redundancy while maintaining predictable decoding and synchronization behavior.

# 3 System Overview

## 3.1 High-Level Encoding and Decoding Pipeline

The system operates in two main stages: encoding and decoding.

- **Encoding Pipeline:**
  1. Read and preprocess input image frames.
  2. Divide each frame into fixed-size macroblocks (e.g., $8 \times 8$).
  3. For I-frames, apply DCT, quantization, zigzag scan, and run-length encoding to each macroblock.
  4. For P-frames, compute the residual with respect to the previous frame, and apply the same compression steps to the residual blocks.
  5. Serialize and write the compressed macroblocks into a binary bytestream.
- **Decoding Pipeline:**
  1. Read and parse the bytestream.
  2. For I-frames, decode each macroblock directly via inverse RLE, inverse zigzag scan, dequantization, and inverse DCT.
  3. For P-frames, reconstruct the residual macroblocks and add them to the previous reconstructed frame.
  4. Reassemble macroblocks into full frame images.

## 3.2 Role of Macroblocks

Macroblocks play a central role in both spatial and temporal compression:

- **Spatial Compression:** Each macroblock is processed independently using the Discrete Cosine Transform and quantization to reduce spatial redundancy within a frame.
- **Temporal Compression:** In P-frames, macroblocks are used to compute and store residuals relative to corresponding macroblocks from previous frames, capturing temporal changes efficiently.

This block-based approach allows for localized processing and is aligned with how real-world codecs like H.264/AVC function, albeit in a simplified form.

## 3.3 GOP Structure Used in the Project

The implemented encoder uses a fixed-length GOP (Group of Pictures) pattern consisting of one I-frame followed by multiple P-frames. For example:

GOP size 5: $I\ P\ P\ P\ P$

GOP size 15: $I\ P\ P\ P\ P\ P\ P\ P\ P\ P\ P\ P\ P\ P\ P$

# 4 Implementation Details

You are provided two helper functions `frame_to_mb.m` and `mb_to_frame.m` which are inverse of each other. You do not have to use them, but they provide a good starting point.

- The function `frame_to_mb.m` accepts a single parameter `frame`. `frame` is expected to have the same shape as the result of the function `imread`.
- The height and the width dimensions of the image is expected to be divisible by 8.
- If the input image is in dimensions (360, 480, 3), the function `frame_to_mb.m` returns a **cell array** data structure `mb_cells` of shape (45, 60) where `mb_cells(i, j)` is the (8, 8, 3) macroblock located at row i and column j.

Implement the encoding and decoding pipeline described in the previous section. Provide two files `decompress.m`, `compress.m` for decoding and encoding respectively. Note that all your functions do not need to be in these files. You can create as many scripts as you want.

- `compress.m` script should encode the video located at `./video_data/` folder we have provided. Resulting **bytestream** should be saved in file `result.bin`. `GOP_size` should be declarable as a constant at the beginning of the script.
- `decompress.m` script should decode the video in file `result.bin` and save the results as `.jpg` files in the folder `./decompressed/` using the same naming convention as the input images. `GOP_size` should be declarable as a constant at the beginning of the script.

You will observe that the total size of the `.jpg` files is much lower than our compressed binary. However, our simplistic approach cannot beat well-established JPEG compression. When we consider the actual video data, with 120 frames and 24 bit for each pixel. The number $480 \times 360 \times 24 \times 120$ is around 62 megabytes. Meaning that the actual data we are compressing is 62 megabytes.

**Important notes:**

- MATLAB imports images as `uint8` by default. We strongly suggest working in type `double`. You can compute everything in `double` and use `uint8` at inital and final steps.
- The use of large language models in an educational manner is allowed. You are expected to be able to explain your code if asked. We strongly advise against directly using code generated by LLMs as this might flag you in our cheating controls.
- In our testing, the video data was compressed into a binary of size 7.8 megabytes using a GOP size of 30. There might be some variance depending on your implementations. However, if you are seeing binaries larger than 30 megabytes, this should be a cause of concern.

# 5 Deliverables

### 5.0.1 Part 1: Simple Algorithm (60 pts)

- (40 pts) Implement the encoding and decoding pipeline described in Section 3. Your submitted code should work for the input files we have provided.
  **Report:** Explain your implementation in the report. Especially, how the encoded data is serialized into a **bytestream** binary should be explained in detail.

- (10 pts) Plot **compression ratio**] with respect to GOP sizes 1 (all I frames) to 30 (I + 29 P frames). Compressed size is the size of your encoded binary. Uncompressed size is $480 \times 360 \times 24 \times 120$ bits.
  **Report:** Include this plot in your report and provide a brief commentary.

- (10 pts) For GOP sizes 1, 15 and 30. Plot **Peak signal-to-noise ratio (PSNR)** curves in a single plot (use colors). Compute PSNR based on MSE between the original frame and the "compressed and decompressed" frame (formula in the link). The x-axis should contain frame number and y-axis should contain PSNR values.
  **Report:** Include this plot in your report and provide a brief commentary.

### 5.0.2 Part 2: Improved Algorithm (40 pts)

- (20 pts) Implement an improved algorithm. Use names `improved_compress.m` and `improved_decompress.m` for this implementation. You **have to** improve based on at least **one of the two** directions:

  1. Add motion estimation to Inter-frame compression (see **Block Matching Algorithm**).
     **Report:** If you choose this direction, explain your implementation and where this algorithm differs from part 1.
  2. Include B-Frames and explore other quantization matrices.
     **Report:** If you choose this direction, explain your implementation, how B-frames work, how you selected your GOP structure, from where did you get quantization matrices etc.

- (10 pts) Same as Part 1, report compression ratios for varying GOP sizes. If your GOP structure do not allow some GOP sizes, you can omit using those values.
  **Report:** Include these values in your report and provide a brief commentary. Compare it with the performance of the simple algorithm.

- (10 pts) Same as Part 1, report PSNR curves. If your improvement involves hyper-parameters other than GOP size, you may also use those as long as you provide a meaningful analysis.
  **Report:** Include these plots in your report and provide a brief commentary. Compare it with the performance of the simple algorithm.

# 6 Submission

Your submissions should include at least

- A report pdf. (Do not forget to include names and student ID's of both group members.) (Only a single submission is sufficient).
- Scripts `compress.m` and `decompress.m` for Part 1 and scripts `improved_compress.m` and `improved_decompress.m` for Part 2. (You may submit more scripts.)
- A README file describing how your scripts should be run, how should the variables are set etc. You may include this in your report too.

Figure files, resulting binaries, input and output images are not needed to be included in the submission.

A very detailed, extensive and lengthy report is not expected. However, a poorly presented and prepared report might result in deduction in points.