Department of Computer Science
Technical University of Cluj-Napoca

JPEG Compression

Laboratory activity 2025

Student: Suciu Andrei
Group: 30431
Email: suciu.se.an@student.utcluj.ro

Image Processing

JPEG Compression

The JPEG Standard

- JPEG is a widely used image compression standard developed during the 1980s and published in 1992 by the **Joint Photographic Experts Group**.
- JPEG is a **lossy** image compression method. It takes advantage of the **DCT** (Discrete Cosine Transform) to encode images.
- It is the most common format for storing and transmitting images the internet.

The JPEG Codec

Although a JPEG file can be encoded in various ways, most commonly it is done with the standard JFIF encoding. This process consists of several steps:

1. The format of the image is converted from **RGB** to $\mathbf{Y'C_bC_r}$ representing **Luminance**, **Chroma Blue**, and **Chroma Red** respectively.
2. The resolution of the chroma channels is reduced, by a factor of 2 or 3. This reflects the fact that the eye is less sensitive to fine color changes than to brightness changes.
3. The channels are split into blocks of 8×8 pixels, and for each block, the **Discrete Cosine Transform** is applied.
4. Each of the blocks is **quantized**, ie. divided using a common **Qunatization Table**, dependent on the quality setting.
5. The resulting blocks are further compressed with a lossless algorithm such as **Run Length Encoding** or **Huffman Encoding**.

Project Overview

- This project implements the JPEG codec for lossy compression/decompression
- This example will not use **Huffman Encoding** for the final compression step
- The project comes with associated headers for reading/writing **.BMP** and the final **.SAMS** binary files
- The program was written for **Linux** systems and must be compiled using the math and threads libraries
- For help, run the command with the `-h|--help` flag

RGB to YCbCr conversion

- **RGB** color space represents colors as combinations of Red, Green, and Blue components
- $\mathbf{Y'C_bC_r}$ separates luminance (brightness) from chrominance (color information)
- This separation allows for more efficient compression, since human vision is less sensitive to chromatic changes

Conversion Formulas:

$$Y = 0.299 \cdot R + 0.587 \cdot G + 0.114 \cdot B \tag{1}$$

$$C_b = -0.169 \cdot R - 0.331 \cdot G + 0.5 \cdot B + 128 \tag{2}$$

$$C_r = 0.5 \cdot R - 0.419 \cdot G - 0.081 \cdot B + 128 \tag{3}$$

Chroma Subsampling

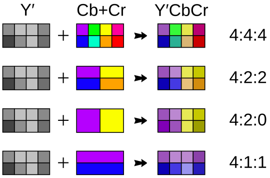

Figure: Chroma Subsampling Examples

- Human sight is less sensitive to color variations than brightness variations
- This allows us to reduce the resolution of the chroma channels without significant quality loss
- The most common subsampling strategy (4:2:0) is to reduce the resolution of the chroma channels by a factor of 2 both horizontally and vertically
- This results in 75% less chroma data

8×8 Block Division

- Each color channel is divided into non-overlapping 8×8 pixel blocks
- If image dimensions are not multiples of 8, padding is applied – for the luminance channel, the previous value is replicated, and for chroma, a neutral 128 is used
- Each block is processed independently
- This approach allows for faster processing and localized compression

Discrete Cosine Transform (DCT)

- DCT converts spatial domain data to frequency domain
- This concentrates block data in fewer coefficients (upper-left corener)
- The DC coefficient (top-left) represents average value of the block
- The AC coefficients (rest of the values) represent frequency components from low to high

2D DCT Formula:

$$\mathrm{DCT}(i,j) = \frac{1}{4} C(i) C(j) \sum_{y=0}^{7} \sum_{x=0}^{7} \mathrm{pixel}(y,x) \cdot \cos\left(\frac{(2y+1)i\pi}{16}\right) \cdot \cos\left(\frac{(2x+1)j\pi}{16}\right)$$

$$\text{where } C(x) = \frac{1}{\sqrt{2}} \text{ if } x = 0, \text{ otherwise } C(x) = 1$$

Quantization Process

- **Most lossy step** in JPEG compression
- A quantization table is calculated according to the desired quality
- Each DCT coefficient is divided by its corresponding quantization table value
- Results are rounded to the nearest integer

Quantization Formula:

$$F_q(i,j) = \text{floor}\left(\frac{DCT(i,j)}{Q(i,j)}\right)$$

Zigzag Scanning

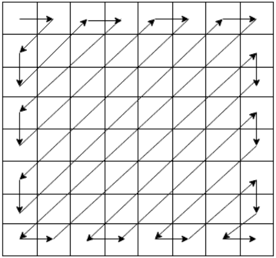

Figure: Zigzag Example

- After quantization, many high-freqency coefficients become zero
- Zigzag scan reorders the 8×8 block into a 1D array
- Scanning pattern groups low-frequency coefficients first
- This creates long runs of zeros at the end of the sequence

Run Length Encoding (RLE)

- Lossless compression step
- RLE encodes zigzag-scanned coefficients
- Represents sequences of zeros folowed by non-zero values
- DC components encoded as difference from previous DC component
- End of block encoded as (0,0)

Example:

Original: [42, 0, 0, 0, 7, 0, 0, 0, 0, 0, 0, 0, ...]
RLE: [(0,42), (3,7), (0,0)]

Decoding Process

JPEG Decoding reverses the encoding steps:

1. Parse the file header to extract **quantization tables**
2. Decode **RLE** encoded data back to coefficient sequences
3. Perform **inverse zigzag** scanning to reconstruct 8×8 blocks
4. Multiply coefficients by their respective **quantization** table values
5. Apply **Inverse DCT** to convert from frequency to spatial domain
6. Restore original resolution for **Chroma channels**
7. Convert from $\mathbf{Y'C_bC_r}$ to **RGB**

DCT Optimization: Separable 1D Transform

Standard 2D DCT vs. Optimized Approach:

- **Standard 2D DCT:** $O(N^4)$ complexity for $N \times N$ block
- **Separable 1D DCT:** $O(N^3)$ complexity - significant speedup!

Precomputed Cosine Table

- Cosine values in DCT formula do not depend on pixel values
- Compute a table with all possible cosine values which could be used in DCT formula

$$\mathrm{DCT}(i,j) = \cos\left(\frac{(2j+1)\cdot i \cdot \pi}{16}\right)$$

Separable Transform Process:

1. **Row Transform:** $R_{DCT}(i,j) = \sum_{k=0}^{7} \mathrm{pixel}(i,k) \cdot C(j) \cdot \mathrm{DCT}(j,k)$
2. **Column Transform:** $\mathrm{DCT}(i,j) = \frac{1}{4} \sum_{k=0}^{7} R_{DCT}(k,j) \cdot C(i) \cdot \mathrm{DCT}(i,k)$

Additional Optimizations

1. Multithreaded Channel Processing

- Each color channel (Y, C_b, C_r) processed in separate thread
- Parallel execution for both encoding and decoding
- **Performance gain:** Up to $3\times$ speedup on multi-core systems
- Implementation: `pthred_create()` for each channel, `pthread_join()` to synchronize before final write

2. In-Place Memory Management

- **Memory type casting:** Same buffer used for float$\rightarrow$int conversion
  - DCT coefficients stored as `float`
  - After quantization, same memory reinterpreted as `int32_t`

Performance Evaluation

Compression Efficiency:

- Acheived 7:1 compression ratio on 24-bit BMP files at 75% quality setting

Processing Speed Analysis:

- **High-resolution image (5184x3456 pixels):**
  - Compression: 0.7 seconds
  - Decompression: 0.5 seconds
- **Low-resolution image (218x241 pixels):**
  - Compression: 0.003 seconds
  - Decompression: 0.001 seconds

Memory Management:

- Memory leak analysis performed using Valgrind
- Zero memory leaks or errors detected
- Peak heap usage: 451 MB for high-resolution image processing

References

1. Wikipedia, JPEG
2. Baeldung, JPEG Compression Explained
3. Stanford University, Lossy Data Compression: JPEG
4. Wikipedia, Discrete Cosine Transform
5. ITU, Studiu Encoding Parameters Of Digital Television, Rec. ITU-R BT.601-7
6. Science Direct, Quantization Table