Universidade de Aveiro

Departamento de Electrónica, Telecomunicações e Informática

Complementos Sobre Linguagens de Programação (2024/25)

Project — Image and Video Compression

Intro

- The project should be implemented in C++. If you are not familiar with C++, start by taking a look of one of the many available tutorials[1].
- The Standard Template Library (STL) will be also a valuable resource[2].
- The comprehensive C++ reference is available at `https://en.cppreference.com`
- Document all the code using the Doxygen tool.
- Use a github (or equivalent) repository to manage the development of your software.

Deliverable I - Image Data Processing

The goal is to work with image data in a similar way to the previous parts on text and audio.

T1 - Reading and loading image files. In C++, you can use libraries like OpenCV (https://opencv.org/) for image manipulation, which makes it easy to read, manipulate, and display images. OpenCV is widely used for computer vision and image processing tasks. Therefore, start by installing this library and look at the tutorials. Note: before trying to download and build the OpenCV library from scratch, find out if there are prebuilt packages for your system.

- Load an image from a file (e.g., PNG or PPM – we will talk about image coding later).
- Display the image using OpenCV functions.

T2 - Visualizing the image channels and the grayscale version of an RGB image.

- Split the RGB image into the corresponding three channels.
- Convert the image from RGB to grayscale.
- Display the color channels and grayscale image and compare it with the original color image.

[1] see, for example, `https://www.tutorialspoint. com/cplusplus/index.htm`

[2] see, for example, `https://www.tutorialspoint.com/cplusplus/cpp_stl_tutorial.htm`

T3 - Calculating image histogram. An image histogram provides a graphical representation of the distribution of pixel intensities.

- Calculate the histogram of pixel intensities for the grayscale image.
- Visualize the histogram to analyse the distribution of intensity values.

T4 - Applying basic image filters (Image filters are used to enhance or extract features from images).

- Apply a Gaussian blur filter to the image with different kernel sizes.
- Display the blurred image and compare it with the original.

T5 - Calculating the difference between two images.

- Implement a function to calculate the absolute difference between two images.
- Compute and display the Mean Squared Error (MSE) and Peak Signal-to-Noise Ratio (PSNR) between the two images.
- Display the difference image to visually observe where the differences are.

T6 - Image quantization. To reduce the number of bits used to represent each image (i.e., to perform uniform scalar quantization).

- Implement the quantization function on grayscale images.
- Experiment with different numbers of quantization levels.
- Compare the original image with the quantized one using MSE and PSNR to evaluate the quality.

Deliverable II – BitStream class

The goal of this task is to develop a reusable and efficient C++ class, BitStream, which will allow for precise manipulation of bits when reading from and writing to files. This BitStream class will serve as a foundational tool in later tasks where efficient bit-level operations are essential, especially in implementing codecs.

T1 - Implementing the BitStream class

You will create a C++ class called BitStream, designed to handle bit-level file operations. This class will be used extensively in subsequent tasks, so it must be optimized for speed and memory efficiency. The BitStream class should enable reading and writing of bits in a file while maintaining the bit order from most significant to least significant within each byte.

The class should implement the following core methods:

1. `writeBit` - Writes a single bit to the file.
2. `readBit` - Reads a single bit from the file.
3. `writeBits` - Writes an integer value represented by N bits to the file, where $0 < N < 64$.
4. `readBits` - Reads an integer value represented by N bits from the file, where $0 < N < 64$.
5. `writeString` - Writes a string of characters to the file as a series of bits.
6. `readString` - Reads a string of characters from the file as a series of bits.

You may add additional methods if necessary, but keep in mind the efficiency of this class, as it will be central to the functionality of future codecs.

Specifications:

- The BitStream class should pack bits into bytes efficiently when writing to a file. When reading, it should extract bits in the correct order from most significant to least significant within each byte.
- Error handling: Ensure appropriate error handling for edge cases, such as reading beyond the end of the file.
- Efficiency: Aim to optimize the class for efficient bitwise operations, as this class will be used extensively in later tasks where performance is critical.

T2 - Testing the BitStream class

The goal of this task is to validate the functionality and performance of the BitStream class developed in Task T1. You will implement a simple encoder and decoder program to convert a text file of binary digits (0s and 1s) into a true binary file, and vice versa. This task serves as a practical test of the BitStream class and will help verify its correctness and efficiency in bit manipulation operations.

Using the BitStream class, implement two programs, encoder and decoder:

1. Encoder: The encoder should take a text file containing only the characters ‘0’ and ‘1’ and convert

this file into a binary file. Each group of eight characters (bits) in the text file should be packed into one byte in the output binary file. The output binary file will thus contain an efficient, compact binary representation of the text file.

2. Decoder: The decoder should take the binary file produced by the encoder and reconstruct the original text file of 0s and 1s. Each byte in the binary file should be converted back into a sequence of eight characters in the text file.

Specifications:

- The encoder should read the text file bit-by-bit, using the BitStream class to write the bits to a binary file.
- The decoder should read the binary file using the BitStream class, extracting bits to recreate the original sequence of 0s and 1s in a text file.
- Padding: If the number of bits in the text file is not a multiple of 8, add 0s to complete the final byte in the binary file. The decoder should ignore any padded bits when reconstructing the original text file.

Deliverable III – Golomb Coding

In this part, you will develop a C++ class for Golomb coding, a type of universal code often used in data compression. Golomb coding is particularly effective for encoding integers with a geometric distribution and can handle both positive and negative values. You will implement a Golomb encoder and decoder using the BitStream class from Part I for bit-level operations. Afterward, you will test the Golomb encoder using a sample text file containing both positive and negative integers.

T1 - Implementing the Golomb coding class

Develop a C++ class for Golomb coding. This class should include functions for encoding and decoding integers, utilizing the BitStream class for efficient bit-level manipulation. The Golomb coding scheme requires a parameter `m`, which controls the encoding length, making it adaptable to specific probability distributions. Your implementation should handle both positive and negative integers. For negative values, provide two encoding approaches (configurable by the user):

1. Sign and magnitude: encode the sign separately from the magnitude.
2. Positive/negative interleaving: Use a zigzag or odd-even mapping to interleave positive and negative values, so all numbers map to non-negative integers.

Specifications:

- Bit efficiency: use the BitStream class for efficient bit manipulation.
- Parameterization: ensure the class can easily switch between modes and values of `m` for flexibility in encoding different distributions.
- Handling negative values: Implement both sign and magnitude and interleaving for representing negative numbers.

T2 - Testing the Golomb Encoder and Decoder

To verify the functionality of the Golomb encoder, you will create a program that reads a list of integers from a text file, encodes each integer using the Golomb class, and then decodes it to confirm that the original values are accurately reconstructed. This test program will use the following steps:

1. Read: open a text file containing integers, both positive and negative, one per line.
2. Encode: use the Golomb encoder to encode each integer, writing the encoded bits to a binary file via the BitStream class.
3. Decode: read the binary file, decode each integer, and verify it matches the original integer in the input file.
4. Output: print a summary of the encoding and decoding results, highlighting any mismatches (if any) and confirming the accuracy of the implementation.

Deliverable IV – Image and Video Coding with Predictive Coding

In this part, you will implement an image and video encoder applying predictive coding techniques for both lossless and lossy encoding.

T1 - Lossless image coding using predictive coding

Implement a lossless image codec for color images using Golomb coding on the prediction residuals. Select appropriate predictors to minimize prediction error, resulting in more efficient Golomb encoding. Your codec should:

- Use spatial prediction to estimate pixel values based on neighboring pixels (ex. JPEG and JPEG-LS predictors).
- Encode the residuals (differences between predicted and actual pixel values) using Golomb coding with optimized values of `m`.

T2 - Intra-Frame Video Coding

Adapt the lossless image codec from Task T1 to support intra-frame video coding, encoding videos as sequences of images. Each frame should be encoded independently (intra-frame coding), using only spatial prediction within each frame.

Ensure that the encoded video file includes all necessary parameters (such as image dimensions and Golomb parameter m) for decoding.

T3 - Inter-Frame Video Coding

Expand the codec from Task T2 to support inter-frame (temporal) prediction with motion compensation. The codec should use intra-frames (I-frames) at regular intervals and inter-frames (P-frames) that encode only changes relative to the previous frame.

Requirements:

1. Intra-frame Periodicity: Let the user specify the interval for I-frames.
2. Block Size and Search Area: Let the user specify the block size and search range for motion estimation.
3. Mode Decision: For each block in P-frames, decide whether to use intra or inter mode based on the resulting bitrate, encoding blocks in intra mode if they result in a lower bitrate.

T4 - Lossy Video Coding with Quantization of Prediction Residuals

Extend the codec to support lossy video coding by introducing quantization on the prediction residuals before applying Golomb coding. Quantization will reduce the precision of residuals, achieving better compression at the cost of some data loss.

Requirements:

- Allow the user to specify the quantization level, which controls the trade-off between compression and quality.

- Quantize the prediction residuals before encoding them with Golomb coding.

Further extensions - DCT-Based Transform Coding

Implement a DCT-based transform coding for lossy video compression. In this task, you will apply the Discrete Cosine Transform (DCT) to each block of a frame, quantize the DCT coefficients, and then encode the coefficients using zig-zag ordering to prioritize low-frequency components.

Steps:

1. DCT transformation: divide each frame into blocks (e.g., 8x8 pixels) and apply the DCT to each block, transforming the spatial data into frequency components.
2. Coefficient quantization: quantize the DCT coefficients to reduce data, preserving more significant low-frequency coefficients while discarding or heavily quantizing high-frequency components.
3. Zig-Zag ordering: use zig-zag ordering on the quantized DCT coefficients, placing the low-frequency coefficients at the start, which tend to carry more important information.
4. BitStream encoding: write the quantized coefficients to a binary file using the BitStream class.

The decoder should rely only on the binary file to reconstruct an approximate version of the original video, reversing the process with inverse quantization and Inverse DCT (IDCT).

Part V – Final Report and Presentation

T1 – Report

Create a report that documents all the significant steps, design decisions, and findings from each task in the project. This report should focus on the outcomes and analysis of your implementation, rather than a detailed description of the code itself.

The report must include:

- Introduction:
  - Briefly describe the purpose of the project and the steps carried out.
  - Summarize the goals of each step (data manipulation, transformation, compression).
- Methodology:
  - Describe the key transformations and algorithms applied.
  - Explain why specific methods were chosen (e.g., why quantization was chosen for image compression or why a certain number of levels were used in quantization).
  - Describe the tools and libraries used in the process.
- Results:
  - Summarize the results of image manipulations, such as histogram calculations, resizing, and quantization.
  - Include MSE and PSNR comparisons between original and processed images to highlight how the various transformations (e.g., quantization, filtering) affected image quality.
  - Include visualizations such as before-and-after images and difference images.
  - Compare the results of your codecs with existing industry-standard codecs (e.g., JPEG for images, H.264 for video).
  -
- Performance analysis:
  - Processing time: measure the encoding and decoding times for various files, discussing any optimizations or bottlenecks encountered.
  - Compression ratios: include compression ratios achieved for each codec, and how they vary depending on the type of data (e.g., image, video).
  - Error metrics: for lossy codecs, calculate the error introduced during compression/decompression (e.g., Mean Squared Error (MSE), Peak Signal-to-Noise Ratio (PSNR) for images).
- Discussion:
  - Analyze the outcomes and identify the strengths and limitations of the methods implemented.

    - Discuss any differences in compression efficiency, quality, and processing time, and analyze why those differences occur.
    - Propose possible improvements or optimizations for future work.
    - Reflect on the learning experience: What was challenging? What worked well?
- Conclusion:
    - Summarize the key takeaways from the project.
    - Highlight what was learned about data manipulation, compression, and image/audio quality analysis.

Specifications:

- Structure: organize the report by sections that correspond to each part of the project, with dedicated sections for performance metrics, analysis, and comparison.
- Clarity: ensure that technical terms and metrics are well-defined, and include visual aids (graphs, tables) where appropriate.
- Brevity: keep descriptions concise and focused on outcomes and analysis rather than code details.

T2 – Final presentation and demonstration

Prepare a presentation and demonstration for the final exam day in June, showcasing the functionality and results of your project.

1 - Presentation outline:

- Introduction: brief overview of the project objectives and methods.
- Highlights of each codec: summarize the approach and key results for each codec (image, video).
- Performance and comparisons: present the key findings on processing time, compression ratio, and error metrics, comparing your results with industry-standard codecs.
- Challenges and solutions: discuss any major challenges faced and the solutions you implemented.
- Future improvements: conclude with ideas for future improvements based on the project outcomes.

2 - Demonstration:

- Demonstrate the working of at least one codec from each category (image and video).
- Show real-time encoding and decoding of sample files, highlighting the achieved compression and, for lossy codecs, the quality degradation.

Specifications:

- Demonstration setup: ensure your software is fully operational and have sample files prepared in advance for the demo.
- Time management: plan to complete the presentation and demonstration within the allocated time (typically around 15-20 minutes).