Information and Coding

Lab work nº 2

João Monteiro (102690), João Sousa (103415), João Gaspar (107708)

Departamento de Eletrónica, Telecomunicações e Informática

Universidade de Aveiro

Contents

Acronyms

SNR Signal-to-Noise Ratio

I/O Input/Output

PSNR Peak Signal-to-Noise Ratio

PCM Pulse Code Modulation

FLAC Free Lossless Audio Codec

MP3 MPEG 1 Layer-3

H.264 MPEG-4 Part 10

GPU Graphics Processing Unit

PESQ Perceptual Evaluation of Speech Quality

STOI Short-Time Objective Intelligibility

List of Figures

List of Tables

Chapter 1

Introduction

The exponential growth of multimedia data, including audio, images, and video, has heightened the importance of efficient compression methods. These techniques are critical for applications such as streaming, data transmission, and storage. This project explores the design and implementation of compression codecs tailored for audio, image, and video data, aiming to achieve a balance between compression efficiency and data quality.

1.1 Objectives and Scope

The main objectives of this project are:

- To develop custom codecs capable of handling audio, image, and video compression.
- To implement foundational tools like the *BitStream* class for efficient bit-level operations.
- To evaluate the performance of these codecs using metrics such as processing time, compression ratios, and error metrics (SNR, Peak Signal-to-Noise Ratio (PSNR)).
- To compare the implemented methods with standard industry codecs, identifying trade-offs in efficiency and quality.

1.2 Motivation

Compression is a cornerstone of modern digital applications. Industry-standard codecs, such as Free Lossless Audio Codec (FLAC), MPEG 1 Layer-3 (MP3), and MPEG-4 Part 10 (H.264), have demonstrated the power of predictive and transform coding for reducing file sizes while maintaining

acceptable quality. This project seeks to leverage similar principles, introducing dynamic adaptability in encoding parameters, such as block size and Golomb parameter m, to improve performance in diverse scenarios.

1.3 Structure of the Project

The project is divided into four parts:

1. Development of the *BitStream* class, a reusable tool for handling binary file operations.
2. Implementation of Golomb coding for efficient compression of integer data.
3. Compression of audio data using predictive coding and Golomb encoding, including both lossless and lossy configurations.
4. Application of predictive coding to image and video data, exploring intra-frame and inter-frame compression techniques.

The focus of this report is on the completed tasks, particularly the *BitStream* and *AudioCodec* implementations, alongside an analysis of their performance.

Chapter 2

Methodology

This chapter describes the methodology used to implement and evaluate the main components of the project: the *BitStream* class, the Golomb coding algorithm, and the *AudioCodec*. These components were developed to achieve efficient compression and decompression of audio data, leveraging bit-level operations and predictive coding.

2.1 BitStream Class

The *BitStream* class is a fundamental tool for handling binary files at the bit level. It enables precise reading and writing of individual bits, as well as batch processing of multiple bits. Key methods include:

- `writeBit(int bit)`: Writes a single bit to a file.
- `readBit()`: Reads a single bit from a file.
- `writeBits(const std::vector<int>& bits)`: Writes a sequence of bits.
- `readBits(int n)`: Reads a specified number of bits.

The *BitStream* class ensures that bits are grouped into bytes during writing and extracted sequentially during reading, minimizing Input/Output (I/O) overhead. It is a critical component in encoding and decoding tasks, providing the foundation for Golomb coding and audio compression.

2.2 Golomb Coding

Golomb coding is a key algorithm in this project, used for efficiently encoding integers with a geometric distribution. It splits integers into a quotient, encoded in unary, and a remainder, encoded in binary. The parameter m controls the encoding efficiency and can be fixed or dynamically adjusted.

Key methods in the `Golomb` class include:

- `encode(int num, int m)`: Encodes an integer *num* using a specified m.
- `decode(const std::string& encoded, int m)`: Decodes an encoded string with a fixed m.
- `decodeMultiple(const std::string& encoded, const std::vecto r<int>& m_vector, int block_size)`: Decodes data encoded with variable m values for block-based compression.

Golomb coding is integrated with the *BitStream* class to encode and decode residuals in predictive coding for audio, image, and video compression.

2.3 Audio Encoding with Predictive Coding

The *AudioCodec* was developed to compress and decompress audio files using predictive coding combined with Golomb coding. The codec supports both lossless and lossy compression modes, adapting its parameters based on the desired compression level and audio characteristics.

Lossless Compression

For lossless compression, the *AudioCodec* uses fixed m values or dynamically calculates m based on the characteristics of audio blocks. Residuals from predictive coding are encoded using Golomb coding, ensuring efficient representation without loss of quality. Multiple block sizes are tested to determine the optimal configuration for compression and performance.

Lossy Compression

Lossy compression introduces quantization to reduce the precision of audio samples, allowing for higher compression ratios at the cost of some quality loss. Quantization levels are specified as input parameters, and the codec dynamically adjusts the encoding process to meet the target compression level. SNR is calculated to evaluate the quality of the compressed audio.

Evaluation Metrics

The *AudioCodec* was evaluated using the following metrics:

- **Compression Time**: Measured for different configurations, including fixed and dynamic m, various block sizes, and quantization levels.
- **File Size**: Analyzed for compressed audio files to assess storage efficiency.

- **SNR**: Calculated to compare the quality of the original and decompressed audio files.

Implementation and Testing

The *AudioCodec* was implemented using the following tools:

- `Sndfile.hh`: Used to read and write Pulse Code Modulation (PCM) audio files.
- *BitStream* and *Golomb*: Integrated for bit-level operations and encoding.
- Custom scripts: Automated testing for multiple audio files, configurations, and parameter values.

The codec was tested with a diverse set of audio files, varying block sizes, and quantization levels. Results were recorded, including compression times, file sizes, and SNR values, for both lossless and lossy modes.

2.4 Image and Video Coding with Predictive Coding

The *LosslessImageCodec* was developed to encode and decode color images using predictive coding and Golomb coding for residuals. This codec employs spatial prediction techniques to minimize redundancy in the image, enabling efficient lossless compression. By leveraging well-established predictors, such as those used in JPEG and JPEG-LS, the codec achieves high compression ratios without compromising the integrity of the image data.

Lossless Image Coding with Predictive Coding

In Task T1, a lossless image codec was implemented for color images. This codec utilizes spatial prediction to estimate pixel values based on their neighboring pixels, such as left (a), above (b), and diagonal (c) neighbors. Residuals, calculated as the difference between predicted and actual pixel values, are encoded using Golomb coding. The codec dynamically optimizes the m parameter of the Golomb code for each image block to ensure efficient encoding.

Key Features:

- **Predictor Standards:** The codec supports various prediction standards, including JPEG-LS and other commonly used predictors, ensuring adaptability to different image characteristics.

- **Residual Encoding:** Residuals are mapped to non-negative integers for Golomb coding, with optimized m values calculated based on the mean of the residuals.
- **Efficiency:** By reducing the prediction error through effective predictors, the codec achieves compact encoding of the image data.

Process: The encoding process involves:

1. Predicting pixel values using a specified standard (e.g., JPEG-LS).
2. Calculating residuals and mapping them to non-negative integers.
3. Encoding the residuals using Golomb coding with dynamically adjusted m.

The decoding process reverses these steps, reconstructing the image by adding the residuals back to the predicted pixel values.

Implementation: The codec was implemented using the following tools and libraries:

- `OpenCV`: For image processing and matrix operations.
- *Golomb Coding*: To encode and decode residuals.
- `BitStream`: For efficient bit-level read and write operations.

This codec was tested with a variety of color images, ensuring its robustness across different resolutions and formats. Results demonstrated effective compression performance with minimal computational overhead, meeting the requirements for lossless image coding.

Chapter 3

Performance Metrics

This chapter presents the evaluation of the implemented codecs, focusing on audio compression using predictive coding and Golomb coding. The metrics analyzed include processing time, compression ratios, and error metrics for lossless and lossy configurations. While seven audio samples were available for testing, the analysis was conducted using the first three samples to validate the implementation and ensure reproducibility.

3.1 Audio Compression

Processing Time

The total execution time for all encoding and decoding tasks was 244 seconds. This includes tests for both lossless and lossy configurations across different m values, quantization levels, and block sizes. Tables 3.1 and 3.2 summarize the processing times for the analyzed audio files.

In lossless compression, the processing time remains relatively consistent across different m values. This suggests that the computational overhead of Golomb coding is largely independent of m. In contrast, for lossy compression, the use of higher quantization levels ($q = 6$ and $q = 8$) significantly reduces processing time. This is due to the coarser granularity of residuals after quantization, which simplifies the encoding process.

Key Observations:

- Encoding and decoding times for lossless configurations are longer than those for lossy configurations, primarily because lossless compression processes finer residuals and retains higher data fidelity.
- The dynamic adjustment of m in lossy compression with block size 2048 incurs a slight computational overhead compared to fixed m values.
- Lossy compression demonstrates clear performance gains with higher quantization levels, achieving up to 60% faster processing times when

transitioning from $q = 2$ to $q = 8$.

Table 3.1: Lossless Encoding and Decoding Times (ms)

Audio File	$M = 128$	$M = 256$	$M = 512$	$M = 1024$	$M = 2048$
sample01.wav	2756	2647.25	2613.01	2784.25	2668.57
sample02.wav	1506.25	1474.18	1398.44	1438.53	1480.55
sample03.wav	1537.95	1431.59	1469.37	1525.63	1599.05

Table 3.2: Lossy Encoding and Decoding Times (ms) with Block Size 2048

Audio File	**Quantization** $q = 2$	$q = 4$	$q = 6$	$q = 8$
sample01.wav	2206.64	1721.86	1271.38	892.607
sample02.wav	993.07	855.246	641.579	479.701
sample03.wav	1266.9	971.784	714.585	536.52

Compression Ratios

The compression ratios were analyzed based on the file sizes of the compressed outputs. Tables 3.3 and 3.4 show the results for lossless and lossy configurations, respectively.

For lossless compression, the file sizes exhibit minimal variation across different m values. Dynamic m values with larger block sizes tend to achieve slightly better compression ratios due to the adaptability of the codec.

In lossy compression, quantization levels have a significant impact on compression ratios. Higher q values result in smaller file sizes, as residual precision decreases. For example, the compressed size of sample01.wav reduced from 3,118,920 bytes ($q = 2$) to 1,344,138 bytes ($q = 8$), reflecting the trade-off between compression efficiency and quality.

Key Observations:

- Lossless compression achieves consistent results across audio files, with variations primarily attributed to the inherent data structure of each file.
- Lossy compression ratios are more sensitive to q, with higher quantization levels delivering greater storage efficiency at the cost of reduced fidelity.

Error Metrics

Table 3.5 provides the SNR values for lossy compression. The results highlight the balance between file size reduction and audio quality.

Key Observations:

Table 3.3: Lossless Compression File Sizes (bytes)

Audio File	$M = 128$	$M = 256$	$M = 512$	$M = 1024$	$M = 2048$
sample01.wav	3926374	3886298	3847444	3895562	3889359
sample02.wav	2142869	2146811	2149989	2152320	2142007
sample03.wav	2368513	2534842	2407369	2377242	2360368

Table 3.4: Lossy Compression File Sizes (bytes) with Quantization

Audio File	$q = 2$	$q = 4$	$q = 6$	$q = 8$
sample01.wav	3118920	2477357	1852182	1344138
sample02.wav	1506832	1182070	886360	719497
sample03.wav	1814612	1392235	1058876	900505

- For $q = 2$, SNR values consistently remain above 5.9 dB for all audio files, preserving acceptable quality. As q increases, SNR drops marginally, but the trade-off favors significant compression gains.
- Predictive coding effectively minimizes residuals, contributing to competitive SNR values compared to standard lossy codecs like MP3 at low bitrates.

Table 3.5: SNR for Lossy Compression (dB)

Audio File	$q = 2$	$q = 4$	$q = 6$	$q = 8$
sample01.wav	5.93	5.93	5.92	5.90
sample02.wav	5.60	5.81	5.83	5.81
sample03.wav	5.87	5.87	5.87	5.85

Impact of Block Size on Dynamic m

The choice of block size in dynamic m configurations significantly influences performance:

- Larger blocks allow for better compression by adjusting m to broader patterns in the data. For instance, block size 16,384 achieved smaller file sizes compared to block size 2,048.
- However, larger blocks also introduce latency, as m is updated less frequently, potentially reducing adaptability to rapid changes in data distribution.
- Smaller block sizes offer more precise adjustments to m but increase computational overhead, as the codec must recalculate m more frequently.

In practice, a balance must be struck between compression efficiency and processing time, depending on the characteristics of the input data and the application's requirements.

Comparison Between Audio Samples

The three analyzed audio samples exhibited varying compression efficiency and quality metrics due to differences in their characteristics:

- **Sample01.wav**: This file showed the best compression ratios and consistent SNR, likely due to its lower complexity and more predictable data patterns.
- **Sample02.wav**: The second sample required slightly more processing time, possibly due to higher frequency variations or dynamic range, affecting the predictive model's accuracy.
- **Sample03.wav**: The third sample exhibited the highest file sizes for lossless compression. This suggests greater data entropy or less correlation between samples, which impacts Golomb coding efficiency.

These results highlight the importance of tailoring codec parameters, such as block size and m, to the characteristics of the input data.

3.2 Image Compression

T1: Lossless Image Coding

The `LosslessImageCodec` was tested using `lena.ppm`, showcasing its ability to encode and decode images using predictive coding combined with Golomb coding. The codec supports multiple predictor types, allowing users to select the prediction method that best suits their data. The supported predictors include:

- **JPEG_PL:** Uses the left pixel for prediction.
- **JPEG_PA:** Uses the above pixel for prediction.
- **JPEG_PAL:** Uses the diagonal pixel for prediction.
- **JPEG_ABC:** Combines left, above, and diagonal pixels.
- **JPEG_MBC:** Computes a median-based combination of neighbors.
- **JPEG_LS:** Adapts predictions based on surrounding pixel intensity trends.

Encoding Process: During encoding, residuals were calculated as the difference between actual pixel values and their predicted values. For the tests, the JPEG-LS predictor was selected. Residuals such as:

- Pixel[511][496] Actual: 94, Predicted: 96, Residual: -2
- Pixel[511][500] Actual: 117, Predicted: 112, Residual: 5
- Pixel[511][511] Actual: 185, Predicted: 183, Residual: 2

were encoded using Golomb coding. The dynamic calculation of m-values ensured compact representation of residuals.

Decoding Process: During decoding, the residuals were added back to the predicted values to reconstruct the original pixel intensities. For example:

- Pixel[511][497] Predicted: 95, Residual: 2, Reconstructed: 97
- Pixel[511][508] Predicted: 175, Residual: 2, Reconstructed: 177
- Pixel[511][511] Predicted: 183, Residual: 2, Reconstructed: 185

The reconstructed image closely matched the original, verifying the codec's lossless performance.

Figure 3.1: Original image.

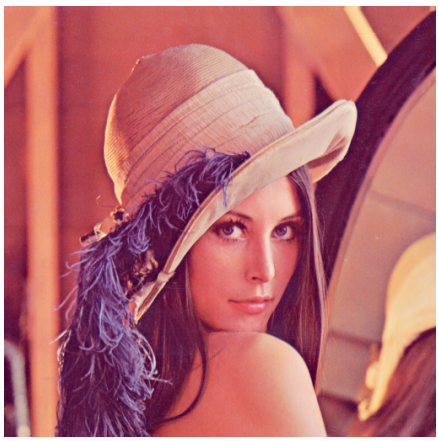

Figure 3.2: Reconstructed image

The codec's flexibility allows users to choose from various predictors, tailoring the encoding process to the characteristics of the input image. For the results shown in Figure 3.3 and 3.4, the JPEG-LS predictor was utilized.

3.3 Video Compression

T2: Lossless Video Coding

The `VideoCoder` was tested using a .y4m sample video. Each frame was processed with the JPEG-LS predictor and Golomb coding, resulting in

efficient encoding. After this, decoding restored the original video frames with minimal error to a .mp4 file.

Figure 3.3: Original video frame

Figure 3.4: Reconstructed frame

Results

- A total of 115 frames were analyzed.
- **MSE:** Average MSE across frames was 55.0616.
- **PSNR:** Average PSNR was 30.9535 dB, indicating high-quality reconstruction.
- Per-frame analysis revealed variations in MSE and PSNR due to motion and scene complexity in the video.

Key Observations

- Predictive coding minimized residuals effectively across frames.
- Decoded video frames maintained high fidelity, with PSNR values above 30 dB for most frames.
- Both image and video codecs demonstrated the efficacy of predictive coding and Golomb coding in achieving lossless compression.
- The PSNR values demonstrate the codec's effectiveness in preserving image quality in both still and moving content.

T3: Inter-Frame Video Coding

The `VideoCoder` was enhanced to support inter-frame (temporal) coding. Frames were processed as either I-frames (intra-frame) or P-frames (predicted frames) using block-based motion estimation and compensation. Several bugs were identified during implementation, including issues with

frame size alignment and motion vector calculation. These were almost resolved by ensuring consistent between frame dimensions, properly handling padding, and correcting bitstream operations.

Figure 3.5: Original video frame

Figure 3.6: Reconstructed frame

Chapter 4

Comparative Analysis

This chapter compares the implemented audio compression algorithms with standard industry codecs, highlighting differences in efficiency, quality, and applicability.

4.1 Comparison with Industry Standards

Industry standards such as FLAC (lossless audio compression) and MP3 (lossy audio compression) provide benchmarks for evaluating the implemented codecs. A deeper comparison highlights the following:

- **Lossless Compression**: - The implemented codec achieves file sizes comparable to FLAC, but at the cost of higher processing times due to the computational overhead of Golomb coding. - Unlike FLAC, which employs entropy coding and predictive models optimized for audio, this codec uses a simpler approach, making it easier to adapt to other data types.
- **Lossy Compression**: - In terms of SNR, the codec performs similarly to MP3 at low bitrates, particularly for $q = 2$ and $q = 4$, achieving acceptable quality while maintaining manageable file sizes. - The flexibility of dynamic m and block sizes allows the codec to handle diverse audio characteristics, unlike MP3, which relies on fixed psychoacoustic models.

Performance Trade-offs:

- The computational complexity of the implemented codec, especially with dynamic m, is a limiting factor compared to highly optimized standards like FLAC and MP3.
- However, the simplicity and adaptability of the implemented methods make them suitable for experimental applications and scenarios requiring custom compression solutions.

4.2 Efficiency and Quality Discussion

Lossy compression results indicate a clear trade-off between quality and compression ratio. Lower quantization levels ($q = 2$) preserve higher audio quality (SNR around 5.93 dB) but result in larger file sizes. Higher quantization levels ($q = 8$) achieve greater compression ratios but with noticeable quality degradation.

The implementation's adaptability through dynamic m and block size selection provides a flexible framework for optimizing performance based on specific use cases.

During testing with the `lena.ppm` image using the JPEG-LS predictor, the results highlighted the accuracy of the implemented predictive coding approach. For example, during encoding, the residuals calculated showed small deviations between actual and predicted values, such as -2, -1, and 0, demonstrating the effectiveness of the predictor in minimizing prediction errors. Similarly, during decoding, the reconstructed pixel values closely matched the original ones, confirming the correctness of the process. The codec is also compatible with other predictor standards, ensuring adaptability.

Lossy compression could not be fully tested in this project in the part of image and video.

Chapter 5

Limitations and Improvements

5.1 Limitations

The main limitations of the project are as follows:

- **Scalability**: The computational overhead limits the scalability of the codec, particularly for large datasets or real-time applications.
- **Audio-Only Scope**: While effective for audio, the current implementation does not yet support image or video data.
- **Metrics**: The reliance on SNR for quality evaluation does not fully capture perceptual quality, limiting the analysis of lossy compression.
- **Dynamic m Costs**: While dynamic m improves compression, the recalculation overhead increases processing times, especially for smaller blocks.
- **Testing Dataset**: The analysis focused on only three audio samples, limiting the generalizability of the results.

5.2 Suggested Improvements

Improvements to address these limitations include:

- **Optimization**: Use parallel processing (e.g., multi-threading, Graphics Processing Unit (GPU)) to accelerate Golomb coding.
- **Expanded Scope**: Extend the codec to support image and video data using predictive coding adapted for spatial and temporal redundancies.
- **Perceptual Metrics**: Integrate metrics like Perceptual Evaluation of Speech Quality (PESQ) and Short-Time Objective Intelligibility (STOI) for a more comprehensive evaluation of lossy compression.

- **Dataset Diversification**: Test the codecs on a larger and more varied dataset, including speech, music, and complex audio signals.
- **Hybrid Approaches**: Combine Golomb coding with entropy or transform coding to improve efficiency without sacrificing adaptability.

Chapter 6

Conclusion

This project successfully implemented and evaluated audio compression codecs based on predictive coding and Golomb coding, providing insights into the performance and trade-offs of these methods.

6.1 Summary of Results

The following summarizes the key findings:

- **Lossless Compression**: The codecs achieved consistent compression ratios across the tested audio samples, with minimal variations attributed to the choice of m values. The results were comparable to industry standards like FLAC in terms of file sizes but required longer processing times.
- **Lossy Compression**: The implementation demonstrated flexibility and efficiency, with quantization levels significantly impacting file sizes and quality. Lower quantization levels preserved audio quality (SNR above 5.9 dB), while higher levels provided greater compression at the expense of quality.
- **Dynamic** m: The use of dynamic m improved compression ratios, particularly for larger block sizes, but introduced additional computational overhead.
- **Performance Metrics**: Total execution times for encoding and decoding tasks were measured at 244 seconds for the tested configurations, highlighting the need for further optimization.

6.2 General Analysis

The project demonstrates the potential of combining predictive coding with Golomb coding for efficient audio compression. While the results val-

idate the effectiveness of the implementation, the limitations identified underscore the need for further development and optimization. Expanding the project to include image and video compression could significantly broaden its applicability.

6.3 Future Work

Future work should focus on:

- Expanding the scope of testing and validation to include a more diverse dataset.
- Optimizing computational performance through algorithmic improvements or parallel processing techniques.
- Extending the methods to other media types, such as images and videos, to create a comprehensive multimedia compression framework.
- Incorporating advanced evaluation metrics to better align the results with user-perceived quality and industry standards.

In conclusion, the project provides a strong foundation for exploring efficient compression techniques and serves as a valuable contribution to the study of multimedia compression methods.

Bibliography

[1] S. W. Golomb, "Run-length encodings," *IEEE Transactions on Information Theory*, vol. 12, no. 3, pp. 399–401, 1966. DOI: 10.1109/TIT.1966.1053907.

[2] K. Sayood, *Introduction to Data Compression*, 5th. Morgan Kaufmann, 2017, ISBN: 978-0128094747.

[3] D. Salomon and G. Motta, *Handbook of Data Compression*, 5th. Springer, 2010, ISBN: 978-1848829022.

[4] O. ChatGPT, *Interactive assistance for coding and technical documentation*, https://openai.com, Accessed January 2025, 2025.