Universidade de Aveiro(UA)

DETI

Information and Coding

Project 2

Eduardo Lopes Fernandes,
João Afonso Ferreira,
Rafael Luís Curado

Teacher: António Neves
Group: P1 - G1

October 2023

Contents

List of Figures

1 Exercise 1 - Extract Color Channel

1.1 Observations

Goal: Implement a program to extract a color channel from an image, reading and writing pixel by pixel, creating a single channel image with the result.

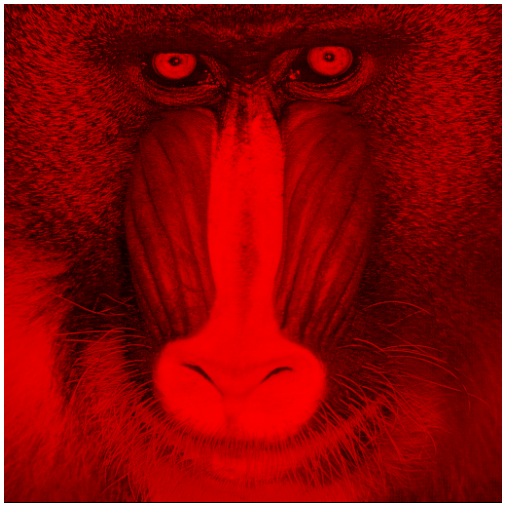
Experiments for exercises 1 and 2 will use the same image to assess accuracy and observe distinctions related to the specified methods. This approach ensures a consistent baseline for comparison, facilitating an examination of the outcomes and variations resulting from the prescribed methods.

For this exercise, it was requested **not** to use the function "split" from **OpenCV** [1]. This C++ program uses the OpenCV library to extract a specific channel from a color image and save the result as a new image. The output is configured to mirror the dimensions of the input image, retaining three channels (**CV_8UC3**) based on user input whether it is 0, 1, or 2 indicating **Red**, **Green**, or **Blue** channel respectively.

To verify the correctness of this program the output images are shown below on Fig. 1. As it is possible to see, the user introduced three different commands for each color channel. This design allows for a clear understanding of the exercise's purpose.

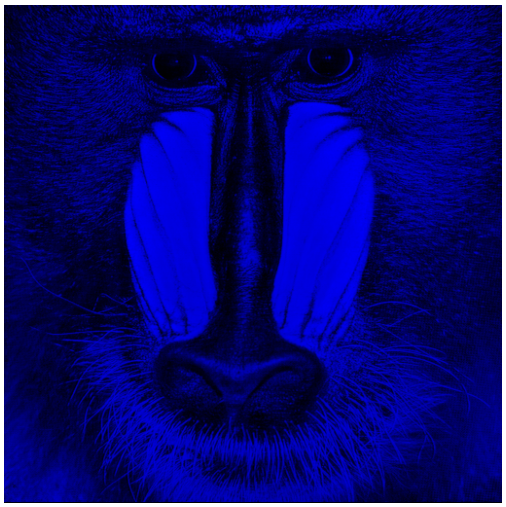
1.2 Results

(a) Original Image



(b) Red Color Extraction

(c) Green Color Extraction



(d) Blue Color Extraction

Figure 1: RGB Extraction Channels

2 Exercise 2 - Image Transformations

2.1 Observations

Goal: Without using possible existing functions of OpenCV, implement programs that:

1. Creates the negative version of an image
2. Creates a mirrored version of an image: (a) horizontally; (b) vertically
3. Rotates an image by a multiple of $90^{\underline{o}}$
4. Increases (more light) / decreases (less light) the intensity values of an image

To enhance code clarity and organization, a decision was made to implement four distinct programs instead of consolidating them into a single entity. This approach aims to minimize confusion and promote a more structured and organized code. That being stated, the programs entitled "negative.cpp", "mirror.cpp", "rotate.cpp" and "light.cpp" were created respectively for each item enumerated before.

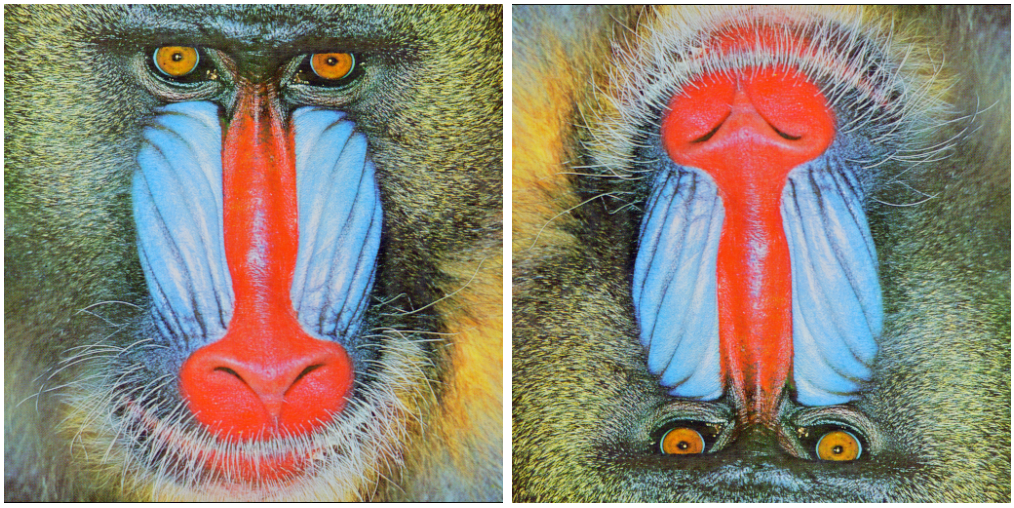
2.1.1 Negative Transformation

(a) Original Image (b) Negative Image

Figure 2: Negative Transformation

The concept of a negative image involves inverting the colors of a given image. Each pixel is represented by three values corresponding to the intensities of Red Green Blue (RGB) channels. In order to create a **negative image**, it is subtracted to each channel the maximum possible intensity value (**255**). This inversion operation transforms dark areas into bright ones and vice versa, producing an image with inverted colors.

2.1.2 Mirror Transformation



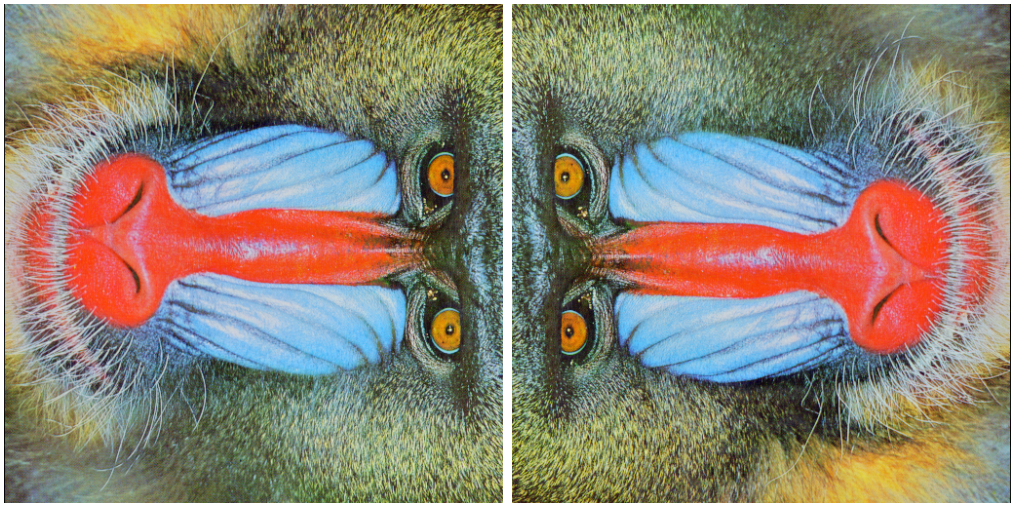
(a) Horizontal Mirrored Image (b) Vertical Mirrored Image

Figure 3: Mirror Transformation

Horizontal Mirroring: In horizontal mirroring, each row of pixels in the image is reversed from left to right. The pixels on the image's left side are swapped with their equivalent pixels on the image's right.

Vertical Mirroring: In vertical mirroring reverses each column of pixels in the image from top to bottom. The pixels at the top of the image with the pixels at the bottom.

2.1.3 Rotate Transformation

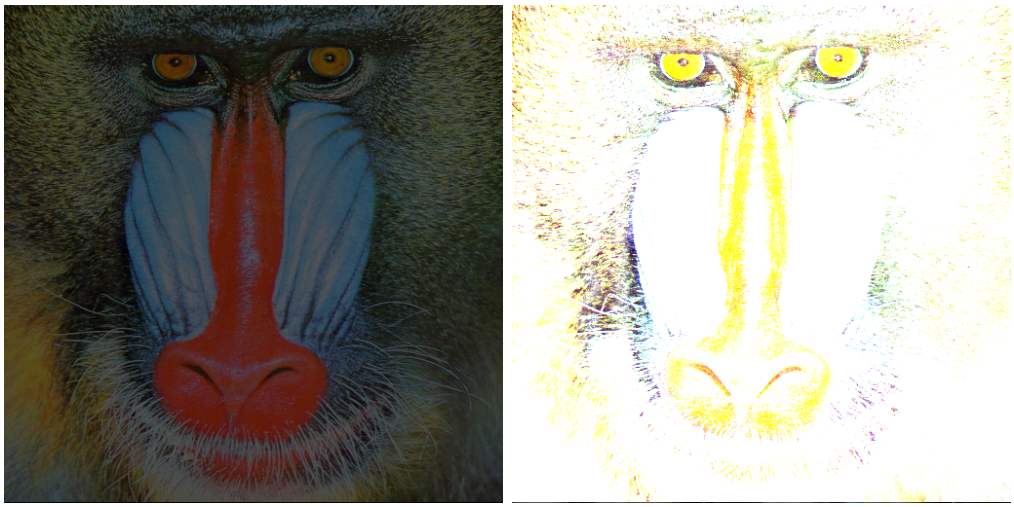


(a) 270 Degrees Rotated Image (b) 90 Degrees Rotated Image

Figure 4: Rotate Transformation

Initially, the program verifies whether the command line argument represents a multiple of 90 degrees. A condition was established where each rotation angle (90/180/270) corresponds to an outcome of 1, 2, or 3, respectively. That being said, the code will perform the rotation operation, alike to those described earlier for mirror transformations.

2.1.4 Brightness Transformation



(a) Reduced Light by 4 (b) Increased Light by 60%

Figure 5: Brightness Transformation

Increasing Light Intensity: Each pixel's color values are multiplied by a user specified intensity factor. If the resulting value is greater than 255 (the maximum pixel intensity in an 8-bit image), it is capped at 255 to prevent overflow. This process effectively brightens the image.

Decreasing Light Intensity: Similar to increasing intensity, the color values of each pixel are multiplied by a factor, but this time the factor is less than 1. If the result is less than or equal to 255, the new value is assigned to the pixel. This operation diminishes the overall brightness of the image.

3 Exercise 3 - Golomb Class

3.1 Observations

Goal: Implement a C++ class for Golomb coding. This class should have (at least) functions for encoding an integer (generating the corresponding sequence of bits) and for decoding an appropriate sequence of bits into an integer..

```
USAGE: ./test_Golomb <file> <0 to write | 1 to read>
```

Golomb coding is a **data compression** technique employing various codes for compression. In order to represent numbers using this coding method, the following procedures are employed [2]:

1. Remainder Representation r: The remainder, r, is encoded in truncated binary, utilizing b bits, where b is determined as the floor of the logarithm base 2 of m ($b = \lfloor \log_2(m) \rfloor$).

2. The remainder, r, will be represented in truncated binary coding, using $b = \lfloor \log_2(m) \rfloor$.

   If $r < 2^b + 1 - m$, code r in binary representation using b bits.

   If $r \geq 2^b + 1 - m$, code the number $r + 2^b + 1 - m$ in binary representation using $b + 1$ bits.

This Golomb class uses the BitStream class for reading and writing bits. The implemented class includes a coding algorithm for encoding and decoding integers.

3.2 Encoding Algorithm

- This encoding method encodes an integer n using Golomb coding and writes the encoded bits to BitStream.
- The algorithm calculates the quotient (q) as well as the remainder r of the division of nn by m.
- Using *unary* coding (writing q ones followed by a zero) it encodes the quotient (q).
- It also determines whether to truncate the binary representation or not based on whether n is greater than or equal to c.
- There is this method that writes the truncated or non-truncated binary representation of r to the BitStream.

3.3 Decoding Algorithm

- This decoding method decodes Golomb-encoded bits from the Bit-Stream and returns the decoded integer.
- It also reads unary-coded bits in order to determine the quotient (q).
- It reads b bits to obtain the remainder (r).
- This method calculates the original integer n based on the quotient and rest.

In order to test the effectiveness of the Golomb class it was implemented a program called "*test_Golomb*".

4 Exercise 4 - Lossless Audio Code

4.1 Observations

Goal: Implement a lossless audio codec, based on Golomb coding of the prediction residuals. The codec should be able to handle both mono and stereo audio files..

For this exercise it was implemented a calculation method for Golomb parameter 'm' based on prediction residuals, encoding audio samples using Golomb coding, and decoding them back to reconstruct the audio (must be the same as the original) [3].

4.2 Encoder

The encoder for the **lossless audio codec** is designed to transform input audio data into a compressed format based on Golomb coding of prediction residuals. The following steps were taken:

Audio File Input: The encoder opens the input audio file (wav file in this case)using the *SndfileHandle* library. It is initialized the necessary parameters, such as sample rate and Block Size (BS). **Prediction Residuals and Histogram:** The algorithm processes the audio file in blocks, calculating prediction residuals for each sample. The residuals are the differences between consecutive samples. Simultaneously, a histogram is created to capture the distribution of these residuals. **Geometric Mean and Golomb Parameter 'm':** The geometric mean of the histogram is important in order to evaluate the predictive effectiveness of the codec. The Golomb parameter 'm' is then derived from this geometric mean. This parameter plays a crucial role in Golomb coding, influencing the efficiency of the compression. Both values were calculated by using the following formulas [4]:

$$p = \frac{n}{\sum_{i=1}^{n} X_i} = \frac{1}{X} \tag{1}$$

$$m = -\lceil \frac{\log_2(1+\mathrm{p})}{\log_2(\mathrm{p})} \rceil \tag{2}$$

Bitstream Initialization and Writing 'm': From BitStream class, implemented in the last project with a slight corrections,is initialized for writing, providing a channel to store the Golomb parameter 'm'. This parameter is written to the bitstream. **Golomb Encoder Initialization and Sample Encoding:** he audio samples are processed once again, and the

prediction residuals are encoded using Golomb coding. The encoded data is written to the bitstream. Concluding, the compressed data is now stored in the Golomb bitstream, ready for decoding.

4.3 Decoder

The decoder reverses the encoder process, reconstructing the original audio data from the Golomb-encoded bitstream. The steps taken for this decoding process will be presented.

Bitstream Reading and 'm' Retrieval: The decoder opens the Golomb bitstream for reading. It gets the Golomb parameter 'm' from the bin file written by the encoder. **Golomb Decoder Initialization:** Using the 'm' value the golomb decoder is initialized in order to decode the encoded data. **Golomb Decoding and Output Audio Sample :** The Golomb decoder reads the encoded data from the bitstream and decodes it. The original audio samples will be rebuild by adding the prediction residuals to the previously decoded samples. Then, the audio samples are written to the output audio file, completing the decoding process. **Output Audio File Initialization:** The expected audio file to be created is to be the same as the original encoding one. (specifying the sample rate, number of channels, and other parameters necessary for writing audio data).

By following these steps, the Golomb encoder and decoder provide indeed a lossless compression solution for audio data using Golomb coding. This dynamic Golomb parameter 'm' ensures efficiency in the compression process.

4.4 Results

To optimize compression with Golomb encoding, we seek an optimal 'm' parameter. Leveraging prediction strategies, we prioritize lower values of 'm' to enhance the efficiency of the Golomb algorithm.

4.4.1 Without Prediction

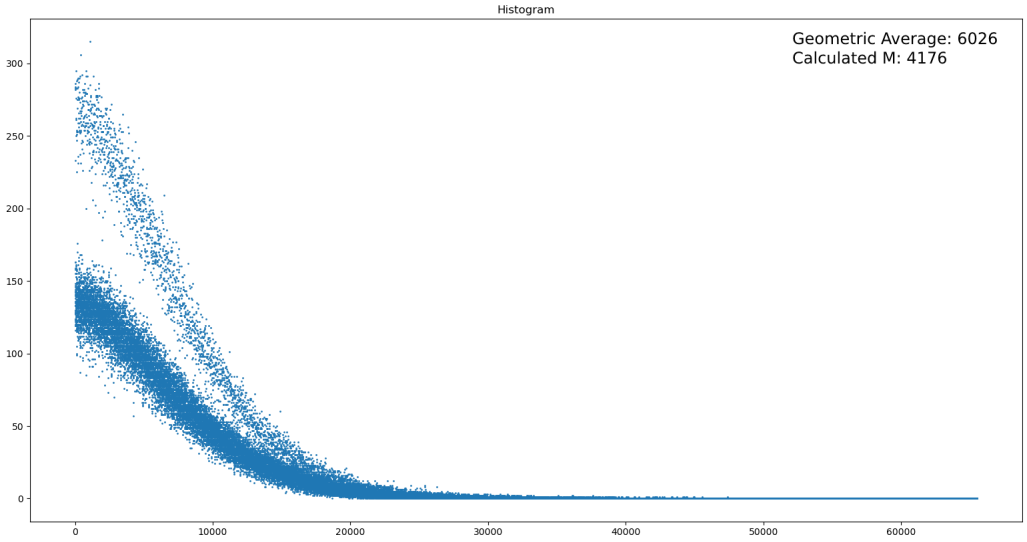


Figure 6: WAV File Histogram Without Prediction

As a foundation, we employed the histogram of the provided audio sample "sample01.wav." Notably, the calculated geometric mean used for determining 'M' is 6026.

4.4.2 Temporal Prediction & Temporal + Inter-Channel Prediction

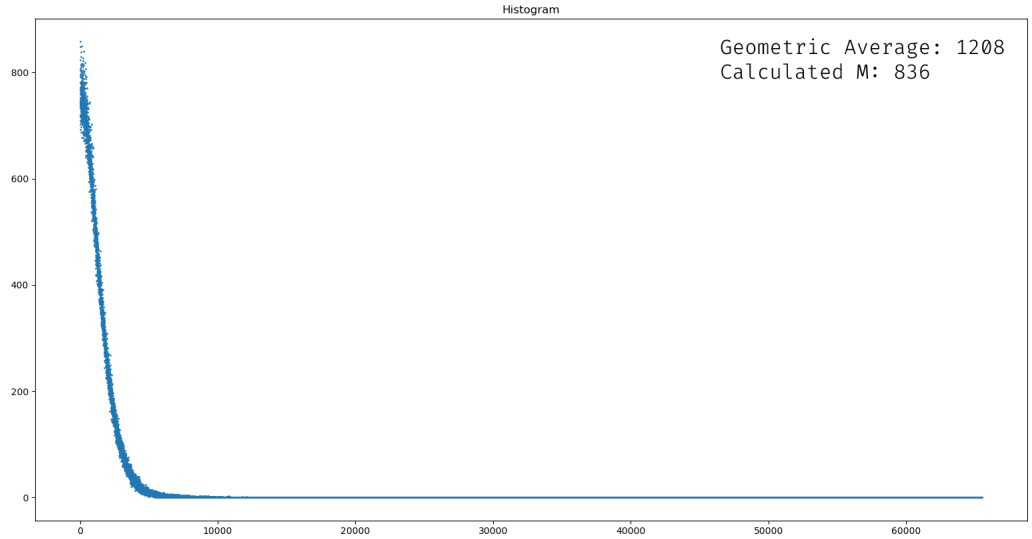


Figure 7: WAV File Histogram With Temporal Prediction

The observed trend indicates a significantly steeper curve, with the average decreasing to 1208. This results in an effective 'M' of 836, which is already quite favorable for our purposes.

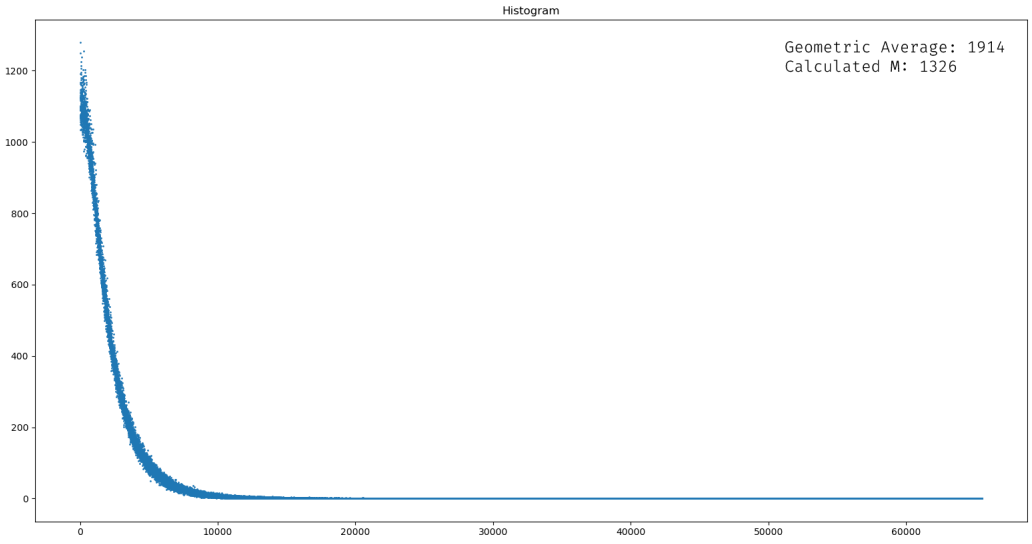


Figure 8: WAV File Histogram With Temporal and Inter-Channel Prediction

Our experiments consistently revealed that inter-channel prediction yielded slightly worse results than the previously employed prediction method. The degree of deterioration varied depending on the WAV file, but across all tested cases, it consistently fell short of the performance achieved with the initial prediction approach.

4.4.3 Encoding and Decoding

Utilizing the determined 'M' value, we successfully encoded and decoded the WAV files. To streamline the process and enhance automation, we incorporated the Golomb parameter storage at the beginning of the encoded file (16 bits).

File	**Original Size**	**Encoded Size**	**M**
sample01	5.2MB	3.8MB	877
sample02	2.6MB	1.9MB	765
sample03	3.5MB	2.4MB	480
sample04	2.4MB	1.7MB	843
sample05	3.6MB	2.4MB	320

Table 1: First Five Samples Encoding Results

5 Exercise 5 - Lossy Audio Codec

5.1 Observations

Goal: to enhance the provided audio coding implementation to support lossy compression, with the average bitrate controlled by a target specified by the user. This involves modifying the existing code to adjust the quantization levels and Golomb parameter to meet the specified bitrate.

Initially, we required a method to compute the bitrate of an encoded WAV file. To address this, we augmented our Golomb class with an additional function that determines the number of bits each value occupies. The bitrate can be calculated using the following equation:

$$bitrate = 44100 * nChannels * \frac{bitsUsed}{totalSamples} \tag{3}$$

Quantization was achieved through a straightforward shift operation, where the 'quant' value signifies the number of bits to be discarded. Notably, quantization precedes the prediction step, ensuring that no additional noise is introduced during this phase.

Having addressed the aforementioned challenges, we can now explore various quantization values and evaluate their corresponding output bitrates. This allows us to identify the optimal quantization value that best suits the requested bitrate.

5.2 Results

```
std::cout << "\nUsage: wavEncode <fileToEncode>\n";
std::cout << "               [ -q quantization (def auto) ]\n";
std::cout << "               [ -m golomb (def auto) ]\n";
std::cout << "               [ -b target bitrate (kbps) (def auto) ]\n\n";
std::cout << "               [ -o output file (def golomb.bin) ]\n\n";
```

Figure 9: Usage of wavEncode.c

The encoder program now seamlessly fulfills both the Ex4 and Ex5 requirements. Users have the flexibility to provide specific Golomb parameters or quantization values. However, if one or both parameters are absent, the program intelligently determines the most suitable values to optimize file compression. In instances where no quantization value or target bitrate is

provided, the encoding process becomes lossless. Lossy encoding occurs only when the user explicitly specifies a target bitrate or a quantization amount.

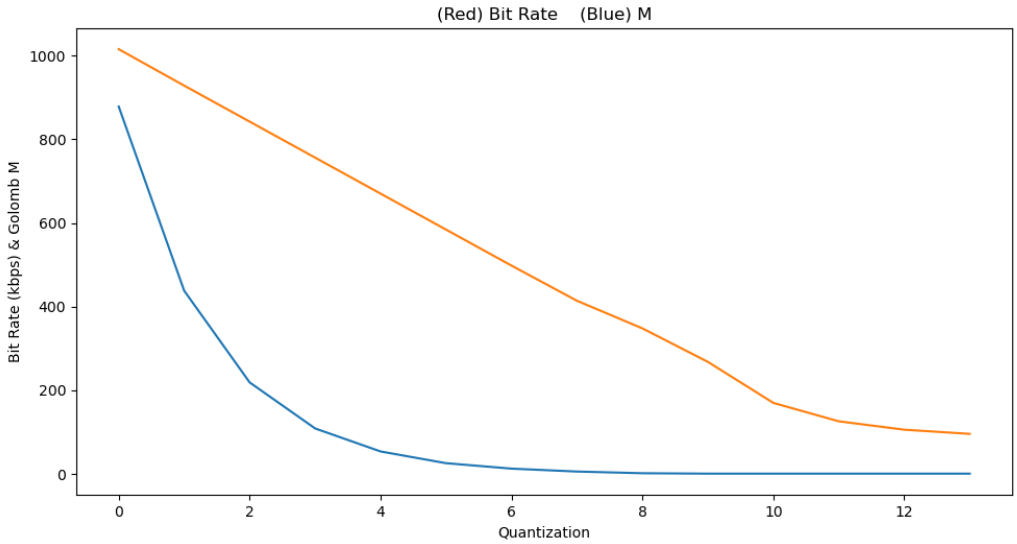


Figure 10: Effect of quantization (used sample01.wav)

In the presented plot, the influence of quantization on both the bit rate and the computed Golomb parameter is evident. The graph illustrates that employing approximately 10 bits of quantization results in a notable reduction of the bitrate by $\frac{1}{5}$. Beyond this threshold, discernible noise and a decline in quality become increasingly apparent.

6 Acronyms

RGB Red Green Blue

BS Block Size

7 Bibliography

References

[1] Tutorials point, *Split images into different channels*, https://www.tutorialspoint.com/how-to-split-images-into-different-channels-in-opencv-using-cplusplus, Accessed: [11/2023], 2023.

[2] K. Zhou, "Lossless audio data compression," PhD/Master's thesis, University NAME, YEAR.

[3] K. Zhou, "Lossless audio data compression," PhD/Master's thesis, University NAME, YEAR.

[4] Ryne Rayburn - projectrhea, *Examples of parameter estimation based on maximum likelihood (mle): The exponential distribution and the geometric distribution*, https://www.projectrhea.org/rhea/index.php/MLE_Examples:_Exponential_and_Geometric_Distributions_Old_Kiwi, Accessed: [11/2023], 2008.