CAV - Codificação Audio e Video Guião 3

Francisco Oliveira (80108), Hugo Sampaio (79961) and Ricardo Chaves (79946)

***Abstract*—The main topic of this guide is video coding. Lossless and lossy encoding techniques were explored. The codecs and auxiliary software were developed using Python 3 and some modules using Cython.**

This report will discuss our process for lossless video coding using both intra coding and hybrid coding and also for lossy video coding. It finishes with the results regarding performance.

I. Golomb Coding

Golomb Coding is a lossless data compression method ideal for situations in which the occurrence of small values is more likely than large values.

This was already a part of the last assignment however some changes were made to accelerate the process of Golomb encoding. These improvements were necessary given the larger number of values to encode in comparison with the audio codecs from the last assignment. To do so, a new Golomb coding module was implemented using Cython that allows to close the performance gap between Python and the C programming language.

II. Extracting and interpreting video frames

The videos to be compressed are all in the YUV color space. This color space has 3 components: the Y, U and V. Y is the luminance (luma) component and both the U and V are the chrominance (chroma) components. These can be sampled in 3 different ways: 4:4:4, 4:2:2 and 4:2:0. For the 4:4:4 sampling there's the same number of luma samples as there is for the chroma components. In 4:2:0 there's half the number of samples for the chroma as there is for the luma component. Finally, for 4:2:0, there's four times the number of samples of luma as there is for the chroma components.

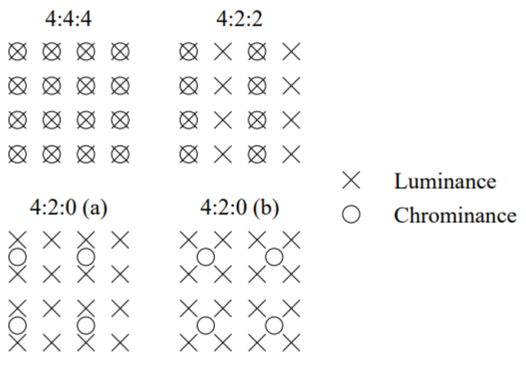

Fig. 1. The sampling methods of YUV images)

To read the information of the video file we need to understand the format of the video file itself. The file has a header, containing information about the file itself, and then the data is divided into a number of frames. Each frame has an array containing the samples of the y, u and v components, first the y array, then the u array and finally the v array (*planar* instead of *packed*. By knowing the sampling scheme and the width and length of each frame in pixels we can calculate the size of each one of these arrays and so we read the exact number of bytes for each one of them and save them. We save them into a object which will contain the 3 components arrays and the width and height of each component matrix. In this object, some methods are provided to access the information, one to access the values directly at a certain position, and another to get a *numpy* array of a desired luma or chroma component.

III. Lossless Intra-frame coding

The goal is to have an encoded file, with information, that can be decoded to retrieve the same information as the original file. This process has no information loss, despite the fact that the encoded file is smaller in size.

To do so, we take advantage of spacial redundancy and variable length codes with Golomb.

It is also relevant to note that every frame in the video is processed by a dedicated process. This allows multiple frames to be encoded simultaneously, while utilizing the multi-processor capabilities of the machine.

A. Applying the Predictor

On this version of the codec, every frame from the video is compressed using intra frame prediction. For this prediction, the JPEG-LS predictor is used and the residuals for each pixel of each frame are retrieved.

B. Residual encoding

Finally, the prediction residuals are encoded using Golomb encoding.

However, to avoid having all the information associated with every frame of the video in memory, the video is processed in batches of frames. For each batch the residual dataset is statistically analysed, values are converted to symbols according to their probability and a ideal M parameter is computed, so that Golomb encoding may be applied.

C. Writing encoded file

The encoded file starts with the same header as the original frame.

As mentioned earlier, there are multiple batches of frames. Each batch has it's own Golomb M values and dictionary, which we will call metadata, followed by the encoded bytes of each channel (planar structure is kept here). As such, a file has the following structure: HEADER\nBATCH BATCH BATCH ...

Each batch has the following structure: \nMETA\n[pickle.dumps(metadata)]\nDATA\nCHANNEL
\n[channel_golomb_data]CHANNEL
\n[channel_golomb_data]CHANNEL
\n[channel_golomb_data]

IV. Lossless hybrid coding

On the hybrid codec some frames are encoded using intra frame coding and other using inter frame coding. To do so, a variable that defines the periodicity of occurrence of a intra coded frame is defined and written to the encoded file.

Every inter coded frame will use as reference frame the last intra coded one.

This encoding process is very similar to the one described in the previous section. The one difference is the process of applying the predictor, however the values that result from this process are also encoded using golomb, just like before.

The process of encoding the inter coded frames has two main steps. First, dividing the frame into blocks. The size of these blocks is a parameter that will be present in the final encoded file or bit stream. Secondly, finding for each block of the frame, the best reference block in the reference frame, that as been divided into blocks in the same way. Finally, computing the residual difference values between the frame block and the reference block from the reference frame.

This process results in two datasets, the motion vectors and the residuals for each block. This are the values that will be encoded using Golomb.

A. Finding reference blocks and Motion vectors

It has been said that for each block of a frame, the best reference block is found in the reference frame. This mean that when coding a block, the blocks from the reference frame must be scanned and compared with the frame block.

This comparison is done by subtracting the matrix that corresponds to the reference block being tested to the matrix that corresponds to the frame block. Then adding the absolute value of every cell of the resulting matrix. This is a measure of the distance between both blocks, and the best reference block is the one that minimizes this difference.

This search process is computationally intensive, so to achieve better performance two strategies where used.

The first one is that the best reference block is found by comparing exclusively the Luma component of each pixel. So the other 2 channels don't contribute to this measure of difference between blocks.

The second one is that not all blocks from the reference frame are considered as candidates to be the best reference block. Only blocks in the neighbourhood of the block being encoded are tested. There is a parameter of the encoder, that specifies the radius in number of blocks that this neighbourhood has, determining the number of blocks that are comprise the search space.

Once the reference block for a certain frame block is found, the motion vector to encode is simply a vector that results from subtracting the frame block's coordinates to the ones of the reference block.

V. Lossy hybrid coding

By introducing lossy coding we start quantizing the prediction residuals of the three color components. For the number of quantization steps we receive 3 values as input, one for each color component. After quantizing the prediction residuals, the coding proceeds the same as if it was lossless hybrid coding.

A. Quantization

Using uniform quantization, we first needed to calculate the quantization step size that we would need to use. Even though we are using uniform quantization our step size couldn't be strictly uniform since we could get a number of quantization steps that wouldn't allow for a uniform split of values through the quantization intervals. Knowing this we decided that we would calculate a interval size that could be uniform for all intervals except the first one, which would be smaller. To calculate the normal interval size we used the following formula:

$$S = ceil(\frac{256}{N}) \tag{1}$$

With N being the number of quantization steps

Following that we would check if the remainder of the integer division between 256 and S was 0. If it was then the quantized value would be calculated with the following formula:

$$Q = floor(\frac{Value}{S}) \tag{2}$$

If the remainder wasn't 0 then we would follow this formula:

$$Q = floor(\frac{Value + (S - (256 \bmod S))}{S}) \tag{3}$$

VI. Decoding

Along side the actual data, some metadata is also present in the encoded file. This included the identification of each frame type, the symbol to value mappings, and Golomb parameters. The decoding process starts by loading this metadata to memory, getting the maps that translate symbols back to value from the file, as well as the M values to be used in the Golomb decoding.

Then, each frame is processed according to its type.

If it is an intra coded frame, the JPEG predictor is used in conjunction with the residual values to retrieve each pixel value of the frame.

However, if the frame is inter coded, the reference frame, the motion vectors and residuals are necessary to decode the frame. After making sure that the reference frame is available, the frame is divided into blocks of the same size used by the encoder and for each block the associated reference block is found by using this block's motion vector. Then a matrix of the same size as this block, containing the residuals for each pixel is applied, achieving the final pixel values of the frame.

After doing this for every encoded frame, the video is decoded.