UCL Université catholique de Louvain

École polytechnique de Louvain (EPL)

Using interframe correlation in a low-latency and lightweight video codec

Dissertation presented by
Guillaume DEMAUDE

for obtaining the Master's degree in
Computer Science Engineering

Supervisor(s)
Benoît MACQ

Reader(s)
Alexandre WILLÈME, Pascal PELLEGRIN

Academic year 2016-2017

Abstract

The resolution of our screens never ceases to increase which means that the data rate of the videostreams increases as well. This becomes a problem for use-cases were uncompressed videostream is used, the bandwidth of some physical links is too limited to allow an uncompressed videostream with those new resolutions. Traditional compression like HEVC can not be used because it is too complex and its latency is too important. This is why lightweight and low-latency compression was created; it allows to compress the videostream in a visually lossless way and its low latency and complexity make it a perfect solution to replace uncompressed videostream. Up until now, most of those low-latency codec do not use interframe correlation to compress the information as it is considered an expensive feature. However, this master thesis explores the way to use interprediction with the constraints of low latency and low complexity in the software implementation of an existing codec: TICO. The new codec created is called ITICO and the results of this software implementation will allow intoPIX S.A. to decide in which use-cases the features are worth implementing in hardware.

Acknowledgements

I am heartily thankful to Pascal Pellegrin, hardware engineer at intoPIX S.A., for his guidance and support throughout this thesis.

I am also grateful to Professor Benoît Macq for his supervision and to Alexandre Willème for his help.

Lastly, I offer my thanks to all of those who supported me in any respect during the completion of the thesis.

Guillaume Demaude

Terms and definitions

AVC	Advanced Video Coding or H.264 or MPEG-4 Part 10 is a block-oriented motion-compensation-based video compression standard.
Bitrate	Number of bits used per pixel. An uncompressed image with a color depth of 8 bits has a bitrate of 24 bits per pixel (bpp). A bitrate of 12 bpp is thus a reduction of the size of the image by a factor 2.
Codec	Short for coder-decoder, algortihms allowing to compress and decompress video.
DCT	Discrete Cosine Transform, operation that decorrelate a signal. Used in older still image compression scheme and still used in video codecs.
DWT	Discrete Wavelet Transform, operation allowing to decorrelate the information of a signal. Commonly used latest still image compression schemes
EBCOT	Embedded Block Coding with Optimal Truncation. Algorithm used in JPEG 2000 that performs the coding and packing of the coefficients of the transformed signal.
FPGA	Field Programmable Gate Arrays. Integrated circuit allowing to be configured after manufacturing.
GCLI	Greatest Coded Trimmed Index. Index of the most significant bitplane of a group of four coefficients.
GCLI	Index of the greatest non null bitplane of a group of for coefficients in a sign-magnitude representation
GTLI	Greatest Line Trimmed Index. Index of the highest bitplane truncated.
GTLI	Index of the highest bitplane truncated for quantization purposes.
HEVC	High Efficiency Video Coding, also known as H.265 and MPEG-H Part 2, is a video compression standard, successor to the widely used AVC (H.264 or MPEG-4 Part 10).
I,B,P frames	Frames treated differently by video coders, I frames are coded using only still image compression schemes. P frames are coded using interprediction.The P frames are coded using only previous P or I frames for the interprediction mechanism. The B frames are coded using previous and following P and I frames.
ITICO	Interprediction Tiny Codec, codec developped during this master thesis based on TICO
JPEG	Joint Photographic Experts Group is a committee dedicated to still image coding standards. Also refers to the first standard of still image compression of the committee.

JPEG 2000	High efficiency still image compression standard.
JPEG XS	Upcoming standard offering a low-latency lightweight image coding system that is able to support increasing resolution.
LUT	Look Up Table. 4-bits input basic component of a FPGA allowing to perform arithmetic operations.
Precinct	2-pixel lines region of a transformed component
PSNR	Peak Signal to Noise Ratio. Measure of resemblance of two pictures, often used to measure the quality of a compression scheme. Its unit is the decibel.
RCT	Reversible Color Transform. Operation allowing to get from one color space to another. Example: RGB to YUV.
RFC	Reference Frame Compression. Compression of the reference frame before its storage.
RGB/YUV	Representation of the colors of a pixel in three components. Red Green Blue, is the basic color space. Y is the luminance, U and V the chrominances. YUV is a decorrelated color space.
SDI/HD-SDI	Serial digital interface (SDI) is a family of digital video interfaces first standardized by SMPTE (The Society of Motion Picture and Television Engineers) in 1989. High-Definition Serial Digital interface (HD-SDI), is standardized in SMPTE 292M; this provides a nominal data rate of 1.485 Gbit/s.
TICO	Tiny Codec, lightweight and low latency codec developped by intoPIX S.A.
VoIP	Video Over Internet Protocol. Broadcast of videostreams using the internet network.

Contents

Introduction

As the resolution of our screens increases so does the data rate of the videos but at the same time, the efficiency of compression algorithms is also improved as attested by the results of the last standard codec HEVC [9].

The concern regarding codecs like HEVC or AVC is their complexity and latency which are not compatible with some use-cases needing ultra low latency. Therefore uncompressed datastreams were used in the past for those use-cases. However, it is not possible to use uncompressed data for 4K or 8K videostreams on SDI/HD-SDI infrastructure or on 1GB-10GB Ethernet links for VoIP.

This is the reason why lightweight and low-latency codecs were created. This kind of compression is also referred to as mezzanine compression and it becomes such an important field that a JPEG Group is currently working on JPEG XS, the new standard of mezzanine compression.

Several codecs of mezzanine compression are currently developed and the main principles of those algorithms are very similar to classical image compression but with a special care for complexity and latency. To that end, the compression is performed image by image without the use of the correlation between the images of a video sequence, as it is considered an expensive feature in term of complexity and a bottleneck in term of latency.

The scope of this master thesis is a software implementation of this aforementioned feature while keeping the complexity and latency relatively low. The development of the codec for this master thesis is based on TICO, originally developed by intoPIX S.A. and the mechanism using the temporality of a sequence is the difference between a reference image and an input image.

In the first chapter, classical compression of image and video are explained with an emphasis on the mechanism that are convenient or not for mezzanine compression. The next chapter defines more properly the concept of mezzanine compression and explains the TICO algorithm. The third chapter describes the modifications added to TICO in order to handle interprediction and shows the results this new codec called ITICO can achieve. A possible lead of improvement is discussed in the fourth chapter and a conclusion closes the thesis.

Chapter 1

State of the art

Before getting into mezzanine compression, it is useful to recall the concepts behind common compression and to examine whether or not the mechanisms used can be adapted to constraints of complexity and latency.

Complexity can be difficult to evaluate. In this chapter, no formal measure of the complexity is used. The general opinion on the complexity of the different algorithms expressed in previous works is trusted. In the next, a measure of complexity is introduced to evaluate the complexity of TICO and ITICO.

Latency on the other hand is easier to observe. It is simply the amount of data that needs to be buffered before the algorithm can start to work on this data.

1.1 Traditional compression

1.1.1 Image coding

The first way to encode a video is to consider it as a sequence of images coded separately with an image compression algorithm such as JPEG or its more recent form: JPEG 2000. This gives the Motion JPEG and Motion JPGEG 2000 codecs.

The main ideas behind most image compression schemes are the same. A first part transforms the pixels that are highly correlated into coefficients with a much lower correlation and with the hope that fewer coefficients are significant for the image; this process is called a decorrelative transform and is often preceded by a color transform which removes the correlation between the Red, Green and Blue (RGB) components of a pixel; the classical color transform with new components YUV is

$$Y = \frac{R + 2G + B}{4}$$
$$U = R - G$$
$$V = B - G$$

The other part consists of the compression of the coefficients resulting from the transform thanks to an entropy coding which is usually lossless (no information is lost) and a quantization which is by definition lossy.

JPEG2000 is the state-of-the-art image coding system released in 6 parts. The first part described hereunder was designed for a relatively low complexity; it was released in December 2000. The figure 1.1 shows the main blocks (some details are omitted like the color transform). The decorrelative transform used in this case is the Discrete Wavelet Transform (DWT). The

coefficients are then scalar-quantized using weights designed to reduce the loss of quality that can be perceived by our eyes. The Embedded Block Coding Optimized with Truncation algorithm is finally used to encode the coefficients.

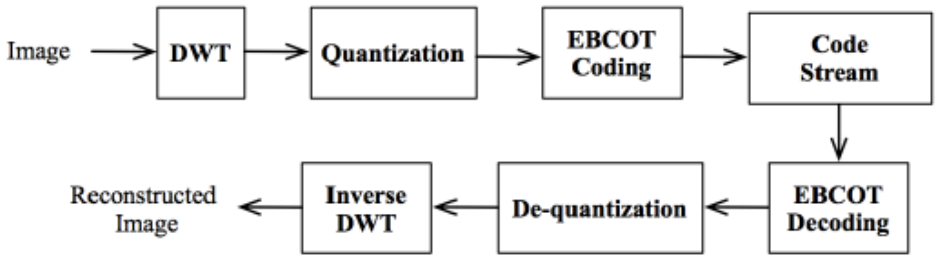


Figure 1.1: Block Schema of JPEG2000 [3]

1.1.2 Discrete Wavelet Transform

As an image is a two dimensions matrix, a 2D-DWT must be used. Happily, a 2D-DWT is obtained by applying 1D-DWT in the horizontal direction and 1D-DWT in the vertical direction thus, only the 1D-DWT needs to be explained.

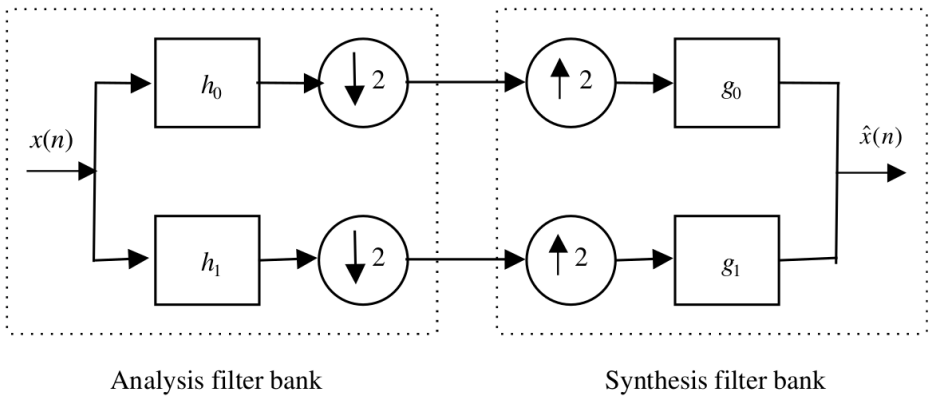


Figure 1.2: 1D-DWT filtering process decomposition and reconstruction of the signal [2]

As shown in figure 1.2, The 1D signal $x(n)$ undergoes a succession of low-pass and high-pass filtering (analysis filter-bank) followed by a down-sampling. The result of the low-pass filters is the image at half the resolution called the low frequency subband, with a loss of details and blurred at each iteration whereas the result of the high-pass filters is the corresponding details required to reconstruct the original image: it is the high frequency subband. The reconstruction is done by up-sampling the coefficients and applying corresponding high-pass and low-pass filters (synthesis filter bank). It is interesting to note that it is not necessary to reconstruct the entire image if the full resolution of the image is not needed, allowing a native multi-resolution compression/decompression scheme.

The figure 1.3 illustrates the application of successive iterations and the subbands resulting of a classical decomposition on a picture. There are many filter banks, some allowing a perfect reconstruction, some of them being better for certain use-cases, some being easier to implement. JPEG2000 uses either the LeGall 5/3 filters implementable with only integers operations (thus is low-complexity) for the lossless/reversible nature of the transform or the Daubechies 9/7 filters for better lossy compression.

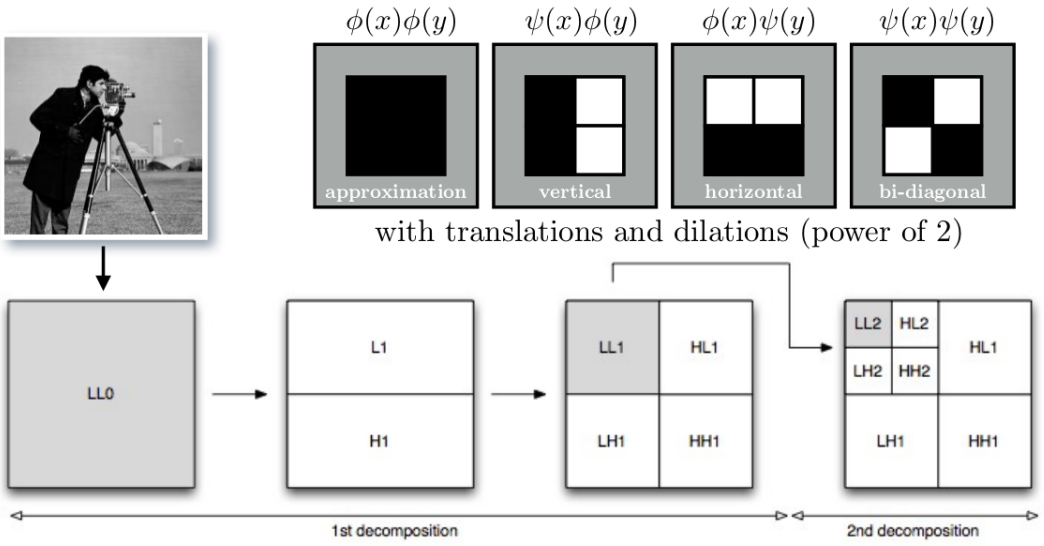


Figure 1.3: DWT process on an image [7]

Applying convolution filters is a complex operation, thus not well suited for real-time applications. Happily, a fast and in-place (the output replace the input in memory) algorithm called the lifting scheme has been found [4]. It reaches the same results as if the image was processed in the way previously explained but reduces the memory needed because only one line must be buffered. The computational complexity is reduced as well. The process is pretty simple, one iteration of the filtering operations is described by the figure 1.4. The coefficients are split into two subsequences according to the parity of their indices. The s^0 and d^0 are respectively the even and odd sequences. After the lifting step, the s^1 are the result of the low-pass filter and the d^1, the high-pass filter. The weights used $(1, -1/2, 1/4)$ depend on the filter bank.

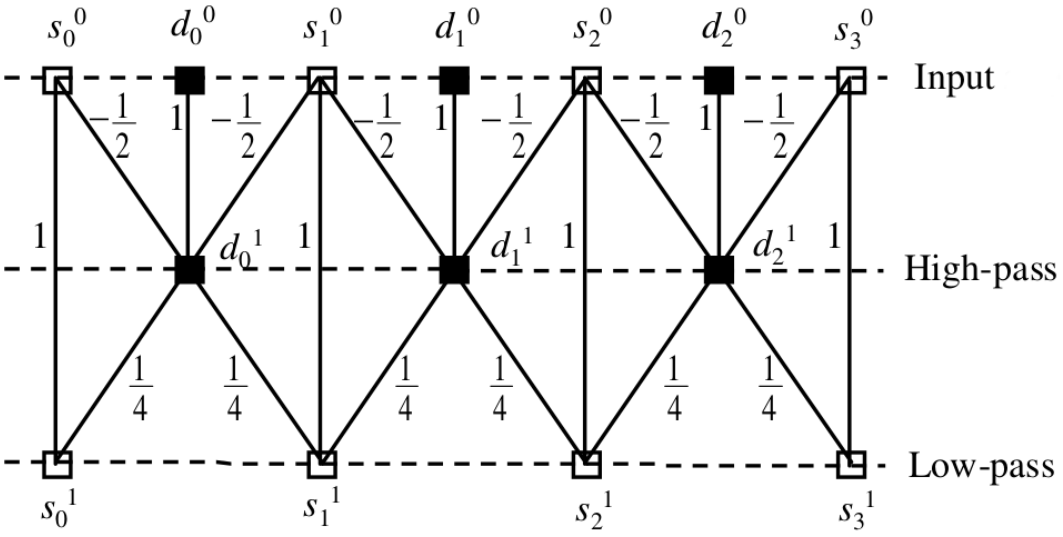


Figure 1.4: Filtering by the lifting scheme [2]

This fast implementation along with the native multi-resolution decompression allowed the Discrete Wavelet Transform to become the main decorrelative transform used in image compression and in mezzanine compression as well.

1.1.3 EBCOT coding

The engine used for entropy coding in JPEG2000 is called Embedded Block Coding with Optimal Truncation (EBCOT). The first step is to quantize the wavelet coefficient and then perform the entropy coding.

The quantization called deadzone uniform quantizer is quite simple. It consists on truncating the Least Significant Bitplanes (LSB) to reduce the data. The mathematical function corresponding is $q(t) = sgn(t)t/d$ with q(t): the quantized value, t: the value of the coefficient and d: the stepsize. A visualisation of the function is shown in figure 1.5. The rate allocation and the perceptual importance of the subband are the factors that determines the step size: the bigger it is, the less accurate the information is. The zone centered on zero is twice as big as the others, it is the deadzone. The dequantizer usually dequantize the q(t) at the middle of the zone. For example, q(t)=1 will become 3d/2. However, the decoder has the choice of the bias it uses.

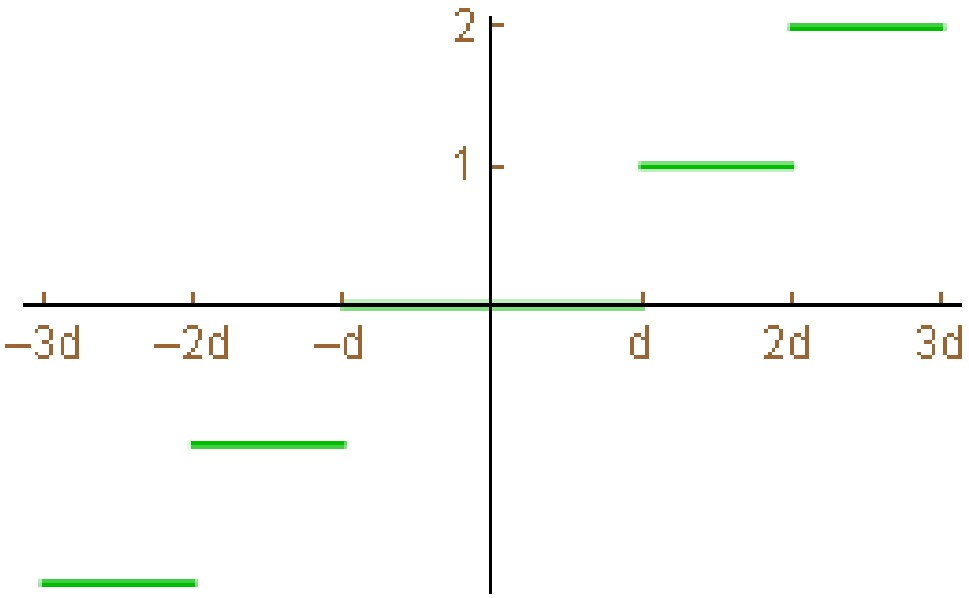


Figure 1.5: $q(t) = sgn(t)t/d$

The quantized indices must be entropy coded. Therefore JPEG2000 made the choice to partition the subband into rectangular codeblocks that will be encoded separately. JPEG2000 uses an adaptive binary arithmetic coder called MQ-coder.

The idea of arithmetic coding is that a sequence of symbols is mapped to a partition interval using the symbols probabilities. The codeword is the binary fraction that points to this interval. Adaptivity is obtained by updating the symbols probabilities using their history. It is important to point out that arithmetic coding is a relatively complex operation compared to other types of entropy coding and that in practical implementation, the probabilities that can be used are limited.

The JPEG2000 algorithm works bitplane by bitplane as shown in figure 1.6. The bits are divided in 3 categories which have 46 probability states each: the most significant non-zero bit of a coefficient is called the significance bit, the previous zero bits are non-significant and the following bits are the refinement bits. Each category is coded by its pass: the Significance Propagation is firstly used to encode the significant bits then comes the Magnitude Refinement for the refinement bits and finally the Normalisation for the non-significant bits. Those operations are the Tier-1 of the EBCOT engine. The Tier-2 is tasked with the organization of the bitstream

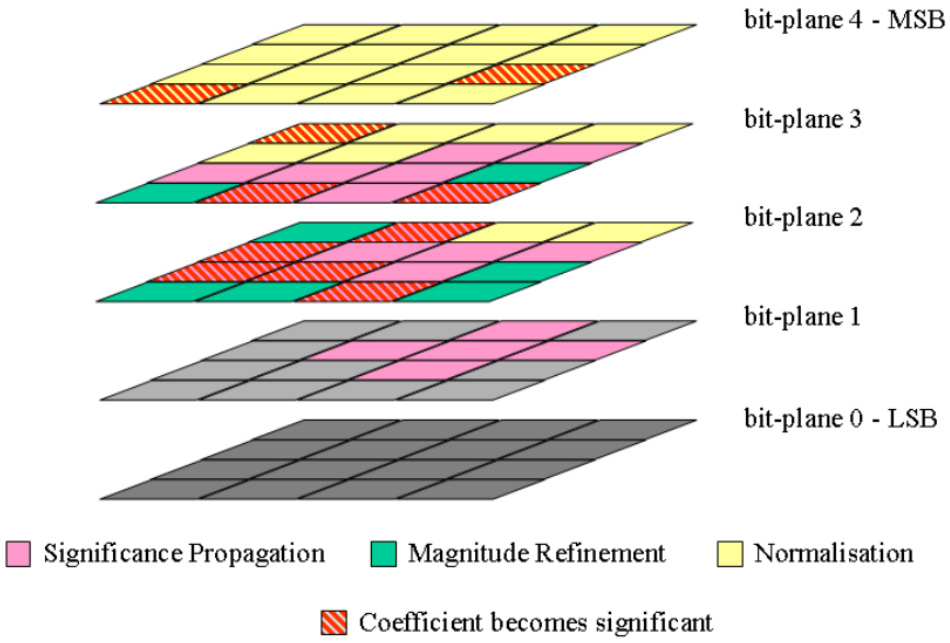


Figure 1.6: Example of coding passes with 5 bit-depth[5]

in order to make its quality and/or resolution scalable and to allow random access. For more details about EBCOT coding, please refer to [5] for an accessible description of JPEG2000 or [2] for an academic article.

The complexity due to several passes of arithmetic coding is not well suited for mezzanine compression which therefore uses simpler entropy coders.

1.2 Hybrid codecs

It quickly appeared that better results could be obtained by taking advantage of the temporal redundancy of a video sequence. Codecs used for this purpose are called hybrid codec as they use intra-frame and inter-frame coding techniques. H.265/HEVC is the latest standard hybrid codec and is slowly replacing H.264/AVC. In this thesis we will only discuss the main steps used in hybrid codecs without entering the details of a particular codec as the basis behind hybrid compression remained the same[1].

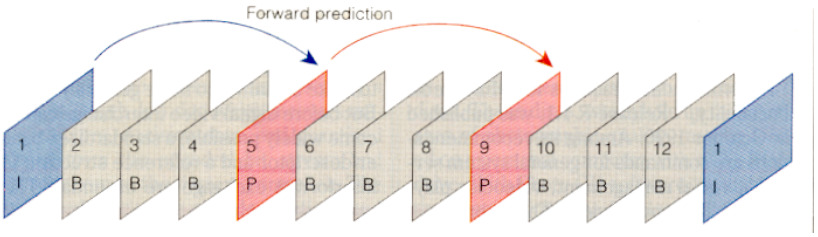


Figure 1.7: Image sequence of MPEG-1 [6]

The first important concept in video coding is that the frames are coded in different ways as shown in figure 1.7 . The I frames are coded using only intra-frame compression schemes. The P frames are coded using only previous P or I frames for the interprediction mechanism. The B frames are coded using previous and following P and I frames. The datastream needs to be reorganized in order to achieve that. The reorganized sequence corresponding to the figure 1.7 is IPBBBPBBBIBBB. In the lastest codecs, a frame is divided into blocks that are marked with the I, P, B flag, making it even more complex to reorganize. This reorganization induces the need of a buffer and an important latency. Such a mechanism is therefore not suitable for mezzanine compression.

The figure 1.8 shows the block schema of a typical hybrid codec. If the input is an I frame, a Discrete Cosine Transform (DCT) is directly applied as well as the quantization and entropy coding to the input. This compression scheme is quite similar to the image compression described above with the exception of the decorrelative transform: the DCT. This transform was previously used in image compression as in the JPEG codec but has been discarded due to the fact it was less efficient than the DWT because working on blocks of the image and not on the entire image. The main problem it may cause is called blocking artifact and can be seen in the sky of the picture 1.9. It was also more difficult to achieve SNR and/or resolution scalability with a DCT than a DWT.

However, DCT is well suited for video compression as the frame needs to be divided into blocks. Indeed, in B and P frames, a step called Motion Estimation and Mode Decision performs an exhaustive search on a predetermined window in order to find a close correspondence between the current block and a block of the previous or following reference frames. The decision of intra or inter-coding a block is generally based on a rate-distortion optimization that requires even more calculations. If a match is found, the information is kept as a Motion Vector that is entropy coded and put in the datastream. The error (difference) between two matching blocks and the blocks that did not find a match undergo the process previously described for the I frames: DCT, quantization and entropy coding.

The exhaustive search performed to find matching blocks between the current frame and several previous or future frames is a real problem in terms of complexity and latency and such a search could not be used in mezzanine compression.

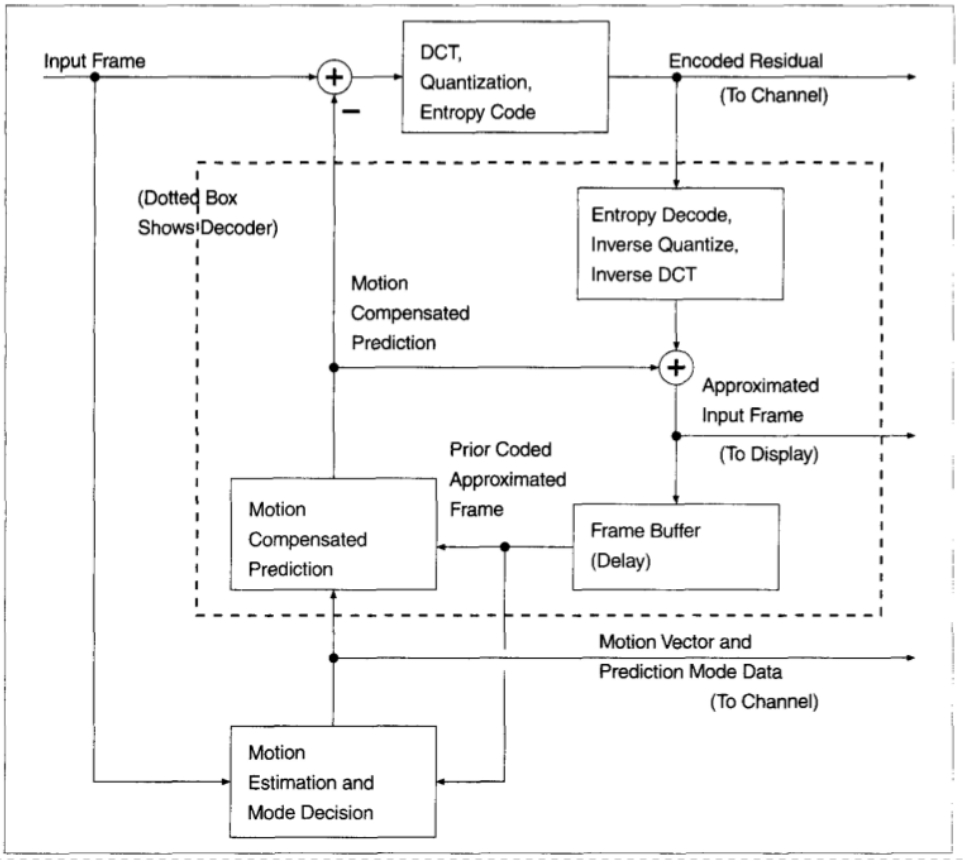


Figure 1.8: Typical video coder[1]

Figure 1.9: Blocking artifacts due to DCT in high compression rate

Chapter 2

Mezzanine compression

Mezzanine compression is based on the same principles as for image compression but is designed with many more constraints[10]:

- Keeping the advantages of uncompressed streams
  - Low-latency: the codec cannot need more than a few lines of the image to work on and produce an output.
  - Low complexity: results in real time for CPU/GPU implementation and low power consumption for physical implementations.
- Reduce the bandwidth required to transmit the stream
  - Visually lossless compression: the data-stream must be reduce by a factor of 2 to 6 without any perceptible loss.
  - Constant/variable bitrate support.

The need for mezzanine compression is quite recent as the video-streams were usually transmitted in an uncompressed form for many cases: professional video links (3G/6G/12G-SDI), IP transport (SMPTE2022 5/6 and proprietary uncompressed RTPs), Ethernet transport (IEEE/AVB), and memory buffers [8]. However, new resolutions like HD, UHD, 4K and soon 8K or even 16K are putting those links under strain. Mezzanine compression is therefore ideal to keep on using those links and to avoid upgrades for more expensive solutions.

Among the professional video links, the 3G-SDI is the most common link used and, as its name indicates, it can only transmit 3Gb/s. A 4K video at 60 fps with a color depth of 8 bits requires 12.75Gb/s for the 4:4:4 format, 8.5Gb/s for the 4:2:2 format thus several 3G-SDI cables are needed or an upgrade to the more costly 12G-SDI cable. A lightweight and low latency video compression with no perceptible degradation of quality could allow to fit the stream within a 3G-SDI connection. The required compression ratio is not too important, the datastream size only needs to be divided by 3 for the 4:2:2 format or by 4.5 for the 4:4:4 format.

There is also a trend toward the use of the Internet Protocol (IP) to broadcast video, the most affordable links for broadcast being the 1G and 10G Ethernet links. Mezzanine compression allows easily to fit a 4K stream into a 10G Ethernet link and can fit a 60 fps Full HD stream (3Gb/s) into a 1G Ethernet link.

2.1 TICO

TICO stands for Tiny Codec and is developed by the company intoPIX. The description of this codec in this master thesis is based on [11] and [12]. Currently, the third version of the codec is

under development with the hope of being integrated into the new JPEG XS standard. TICO follows the constraints described in the previous paragraph with the particularity that it only supports constant bitrates and achieves a latency of only a few lines in hardware implementation and a single frame latency in software implementation.

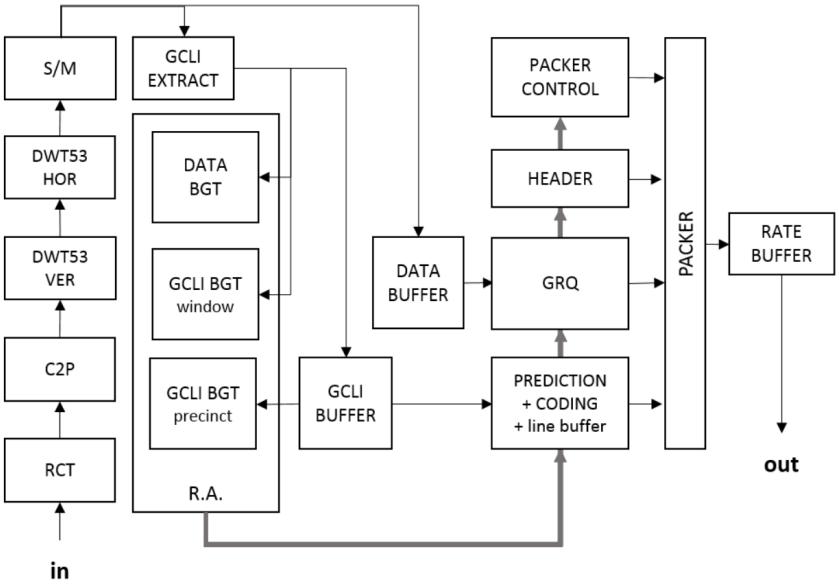


Figure 2.1: Global blocks decomposition of TICO encoder [12]

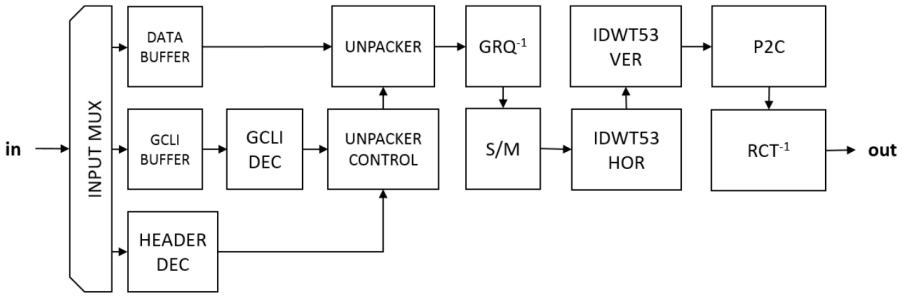


Figure 2.2: Global blocks decomposition of TICO decoder [12]

The block diagrams of the TICO coder and decoder are shown in figures 2.1 and 2.2.Here is the description of the main steps of the codec:

- The first block (RCT) is a classical color transform only performed if the input is in a RGB form: it processes the three components 2 pixels in parallel. The following blocks are the decorrelative transform which is a DWT with the particularity of having a special decomposition. There is only one vertical decomposition, then five horizontal decompositions for the low vertical frequency and one horizontal decomposition for the high vertical frequency (see figure 2.3). The filter bank used is the LeGall 5/3 as it allows to use the lifting scheme described in chapter 1 with only integers operations. The depth of the resulting coefficients is 16 bits. The coefficients are converted in a sign magnitude form for the entropy coder. The DWT processes the 3 components of 4 samples in parallel.
- The rate allocation block (R.A) decides how far the quantization must go depending on the bitrate budget on a window of typically 8 lines, the efficiency of the entropy coding and the importance of the subband (predetermined weights).

- The coefficients undergo the entropy coding (GCLI prediction coding) and the quantization.
- Lastly, the packer organizes the datastream.

TICO complexity

Processing Unit	Memory	Complexity in LUTs
Encoder		
RCT	0	250
Vertical DWT	2 lines	1000
Horizontal DWT	Half a line	2000
Data buffer	10 lines	100
GCLI coding	less than 8kB	700
Rate allocation	0	2000
Quantization	0	800
Data Packing	0	1200
Rate smoothing buffer	14 compressed lines	100
TOTAL	380kB	8150*1.5= 12000[1]
TOTAL + filtering improvement	380kB	12000+3000= 15000
Decoder		
Input Mux	0	200
Data buffer	14 compressed lines	100
GCLI buffer	14 lines of RAW GCLI (19kB)	100
GCLI decoding	less than 8kB	1100
Data unpacking	0	1200
Quantization^{-1}	0	1400
Horizontal DWT	Half a line	2000
Vertical DWT	2 lines	1000
RCT^{-1}	0	250
TOTAL	180kB	7350*1.5= 11000
TOTAL + filtering improvement	180kB	11000+2500= 13500

Table 2.1: Complexity and memory of each processing unit in TICO for video 4K 60fps and hardware frequency at 300MHz for the DWT and 225MHz for the packing blocks

The complexity of the TICO coder and decoder shown in figures 2.1 and 2.2 is evaluated thanks to one main criteria: the number of Lookup Tables[2](LUTs) of each module. The FPGA memory needed for each module is also taken into account. The complexity estimation is made for a video sequence at a 4K resolution at 60 frame per second and without downsampling of the chroma components. The clock frequency of the codec is at 300 MHz for the DWT and 225MHz for the packing blocks. The complexity estimation and memory required for each module is shown in table 2.1, the data is taken from [12]. The total estimated number of LUTs is 12000 for the coder and 380kB of buffer are needed. The decoder is simpler than the coder as all the information needed to decode the stream is in the headers thus only 11000 LUTs and 180 KB of buffer are needed. An improvement of the codec discussed later increases this complexity from 12000 and 11000 to 15000 and 13500 LUTs.

[1]Multiplication factor accounts for control operations, integration, etc.
[2]The 4-bit input Lookup Table (LUT) is the basic component of a FPGA to perform logical operations

TICO latency

Encoder	
Module	Latency in number of lines
Vertical DWT	2
Horizontal DWT	1/2
Rate allocation	8
Rate Buffer	8
GCLI packing	1
TOTAL	20
Decoder	
Rate Buffer	8
GCLI unpacking	1
Horizontal DWT	1/2
Vertical DWT	2
TOTAL	12

Table 2.2: Latency of TICO encoder and decoder

The hardware latency of the different modules is presented in table 2.2. As expected, the total latency is very low.

The parts of TICO described in this chapter are the entropy coding, the quantization process, the rate allocation process and the packer as they are specific to this codec.

2.1.1 Entropy coding and quantization

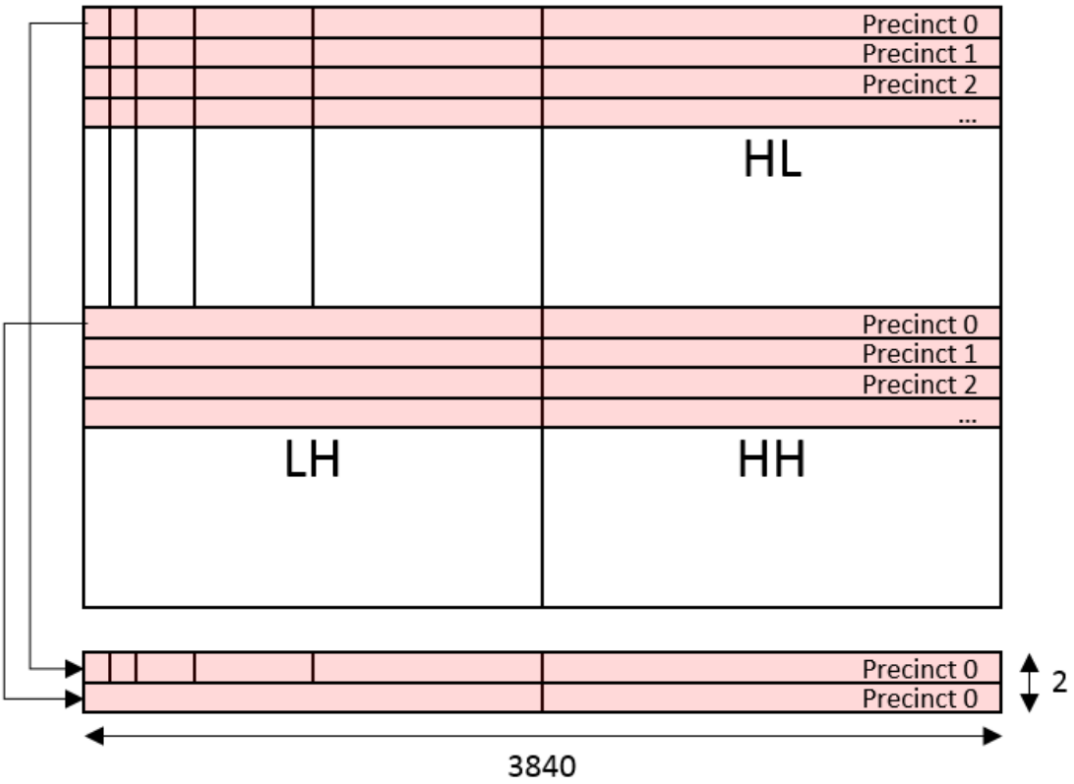


Figure 2.3: Representation of a 3840x2160 frame after a DWT [12]

Figure 2.3 shows the state of a frame after the wavelet transform. The data is organized by precinct, a 2 lines structures. The content of a precinct is composed of 8 subbands by component. The subbands coefficients are grouped into subsets of 4 samples. For each group, the Greatest Coded Line Index (GCLI) is computed and the Greatest Line Trimmed Index (GTLI) is given for a subband. They are respectively a pointer to the most significant non-null bitplane of this group excluding the sign bit and a pointer indicating the bitplanes that are truncated due to quantification. The GTLI is decided by the rate allocation mechanism. Those pointers allow to send to the output stream only the relevant bitplanes. The GTLI must not be sent as it can be computed from information in the picture header (weights of the subbands) and information in the precinct header(truncation scenario). The GCLI's sent in the data flow are the "truncated" GCLI's: the new position of the GCLI's after truncation of the bitplanes indicated by the GTLI. If the GTLI is greater than the GCLI, no bitplanes are sent and the truncated GCLI is set to 0.

The new values of the coefficients after truncation of the bitplanes indicated by the GCLI and GTLI can be mathematically described by the following formula, x being the coefficient value and y the quantized value:

$$y = \lfloor (x >> gtli) - (x >> (gcli + 1)) + 1/2 \rfloor$$

This formula describes a method called Center-rounding quantization. The more classical Deadzone-Uniform method is available as well but is less efficient than Center-rounding.

Figure 2.4 illustrates with a 8 bits depth example but, in reality, coefficients have 16 bits depth. In this example, the GCLI is, starting from 0, in the fourth position thus the GCLI is equal to 4 and the GTLI is equal to 1. This means that only the bitplanes in position 1, 2, 3 and the sign bitplane are sent along with the truncated GCLI. The quantized values (using deadzone quantization) of the example are -2, -7,3,-2 and the GCLI sent is 3.

	-5	15	6	-4
Sign →	1	0	0	1
	0	0	0	0
	0	0	0	0
GCLI →	0	0	0	0
	0	1	0	0
	1	1	1	1
GTLI →	0	1	1	0
	1 (cut)	1(cut)	0(cut)	0(cut)

Figure 2.4: GCLI and GTLI on a four coefficients group

In order to improve the lossless compression, the "truncated" GCLI's are actually compressed thanks to a prediction mechanism. There are two prediction modes to encode the GCLIS.

- Vertical Prediction: the result is the difference between the GCLI of the current and the GCLI of the same subset of coefficients in the previously coded precinct. The first precinct

of each slice can not use this mode.

- Horizontal Prediction: the result is the difference between a GCLI and the previous GCLI of the same subband. In that mode, the first GCLI of each subband must remain unmodified.

If the GCLI's are unmodified, they have a depth of four bits whereas the predictive value are Unary Encoded with codewords going from one to fifteen bits. Therefore, both predictions are tested as well as leaving the GCLI unmodified and the most efficient method is chosen for each precinct. The kind of coding is put in the precinct header.

The decoding is straightforward: thanks to the header of the precincts and of the frame, the decoder has enough information to compute the GTLI's and to decompress the GCLI's. The dequantized values using Center-rounding dequantization, x', are given by this formula:

$$x' = \lfloor y * \frac{1 << (gcli + 1)}{(1 << (gcli - gtli + 1)) - 1} \rfloor$$

2.1.2 Rate allocation

Basic mechanism

An encoded precinct contains 3 parts: the header, the encoded "truncated" GCLI's and the raw bit-planes data. As the rate allocation mechanism determines the GTLI's, it can modify the size of the of raw bit-planes data but also the encoded GCLI's. Indeed, If the GTLI is greater than the GCLI, the truncated GCLI is set to 0 and the difference between adjacent GCLI's is reduced thus the prediction gets better.

In order to reach a determined precinct budget that can simply be the average precinct budget, an iterative process is executed.

$$AverageBudget = \frac{CompressedImageSize}{NumberPrecincts}$$

The first phase of this process, called Scenario consists of computing the GTLI of the subbands at a scenario iteration, i, taking into account the weight of the subbands, gain[subband]:

$$GTLI(subband)_i = MAX(0, i - gain[subband])$$

For each iteration, the raw data-bit size is evaluated knowing the GTLI's and the GCLI's and the size of the encoded "truncated" GCLI's are computed as well. If the computed size is below the precinct budget, this phase stops. Otherwise, it continues with i+1.

After the Scenario comes the Refinement phase: re-adding one bitplane per subband in the order defined by a priority table until it exceeds the budget again. The gain table and the priority table are predetermined weights in a setting. There are two main sets of weights: weights for visual optimization and weights for PSNR optimization.

Quality smoothing

An ideal rate allocation would consider an entire image and allocate a more important bitrate on difficult parts of the image but this would be too complex and would require to buffer the entire image.

Instead, TICO uses a sliding window of four precincts. The first step is to execute the Scenario phase on those 4 precincts at once with a budget of four times the average budget:

$B_W = 4B_A$. After this step, each precinct has a budget (b_0,b_1,b_2,b_3) and there may be left over bytes(lb).

The second step executes the Scenario-Refinement process on the first precinct of the window with a budget of $b_0 + lb/4$.

The window shifts one precinct below and the first step starts again using a budget of $B_W = 4B_A + R = b_1 + b_2 + b_3 + 3lb/4 + B_A$ with R the reported budget.

In order to limit the latency, the reported budget, R, can not be reported too far and therefore a limit to the reported budget of six times the average precinct budget called R_{max} is set. If the reported budget exceeds R_{max}, the first precinct of the window get a bigger budget for its rate allocation and some padding is added.

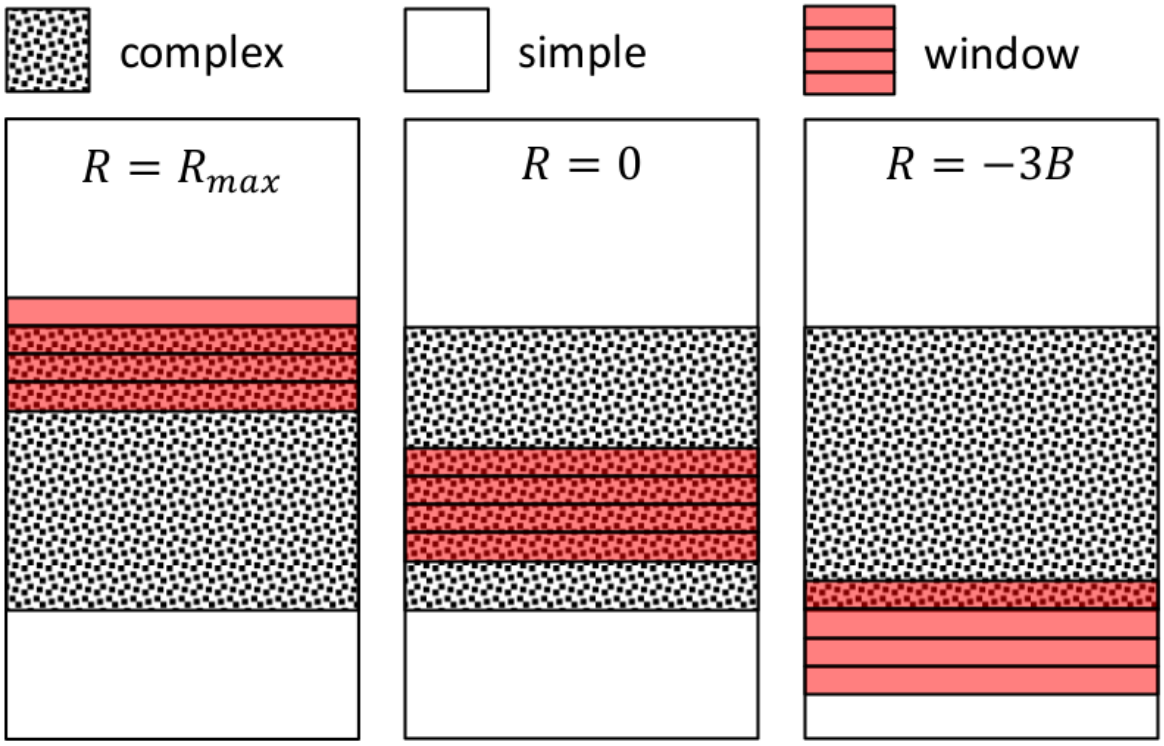


Figure 2.5: Example of rate allocation process [12]

Figure 2.5 shows an example of the quality smoothing process in an simplistic way. When the window is in the first simple area, the precincts do not need a big budget thus the maximum reported budget is reached. Then comes the complex area: the reported budget is rapidly consumed and gets to 0 as every precinct consumes the maximum budget available. Once the window gets out of the complex area, the complex precinct takes all the budget of the window given that the simple precincts do not need it: $b_0 = 4B_A$, $b_1 = b_2 = b_3 = 0$ thus when the window slides down, $B_W = 0 + 0 + 0 + B_A = 4B_A + R$, which means that the reported budget: $R = -3B_A$. The four simple precincts of the new window will use only 1B as window budget.

2.1.3 Packer

The first role of the output stream packer is to receive the information from the other blocks and to organize all the data: picture header, slice headers, precinct headers, data bit-planes and GCLI's. If the packer plays this sole role then the complexity is 12000 LUTs.

However, to improve the quality of the compression, some new features were implemented in the third version.

The first one is the filtering of sign bits of null coefficients. Indeed, if a coefficient was null within its GCLI group, its bit sign was part of the raw data unless the GCLI was in the 0 position. As the coefficient is null, its sign is useless information, and therefore filtering it improves the efficiency. In addition to the filtering mechanism, a reorganization of the packing is required and the rate allocation is adapted to take this filtering into account when estimating the budgets by getting the Most Significant Bitplane (MSB) of each coefficient as additional information.

A second filtering is performed to remove the GCLI's that are null and therefore not useful. Indeed, due to the nature of the wavelet transform and to the quantization, a great amount of GCLI's are null. The filtering is applied on groups of eight GCLI's within a same subband, if all are null then they are removed and a bit flag is set to 1. The choice of groups of eight GCLI's for the filtering was an trade-off between the bits spared thanks to the filtering and the signalization bits added. Again, the rate allocation takes into account this filtering when estimating the budgets.

Those two mechanisms increase the complexity of the coder and decoder from 12000 and 11000 LUTs to 15000 and 13500 respectively but the general opinion was that the gain was worth it especially at low bitrates.

Chapter 3

ITICO

The main idea behind this master thesis is to find a way to use interframe correlation with a relatively low increase of complexity and latency and a buffer as small as possible. The purpose of the "Interprediction Tiny Codec", ITICO, is the compression of desktop video sequences. Indeed, TICO compression in low bitrates, 1-2 bits per pixel (bpp), of desktop images results in visible degradation as the DWT is not adapted to compress text on images. Using interprediction may remedy the problem.

In the scope of this master thesis, a software implementation of ITICO is developed on the basis of the C code of TICO. The constraints of latency, hardware complexity and memory are taken into account during the development for an hardware implementation.

3.1 Main concept

Description

To remain within low latency and lightweight constraints, the main mechanism chosen is a matrix difference between an input frame and a reference frame kept inside a buffer. This difference is then normally encoded with TICO. The decoder receives the encoded difference, decodes it as a normal frame and adds it to the reference frame that becomes the new reference frame as well as the output frame as shown in figure 3.2. In order to avoid a drift between the reference frame of the encoder and the decoder, the dequantization and the inverse transform must also be performed at the encoder as shown in figure 3.1.

Thanks to this scheme, a sequence of still images or with few movements should have gradually less distortion. Indeed, the first frame is compressed using a normal TICO. The reference frame is therefore a lossy version of the first frame. Then comes the second frame, identical to the first. The difference between the second frame and the reference frame corresponds to the loss of information due to quantization in the first frame. This difference is compressed and then added to the reference frame which therefore recovers a part of the information that has been lost. And this process continues until there is no more information to be retrieved: the output frame is therefore a perfectly identical image to the input image.

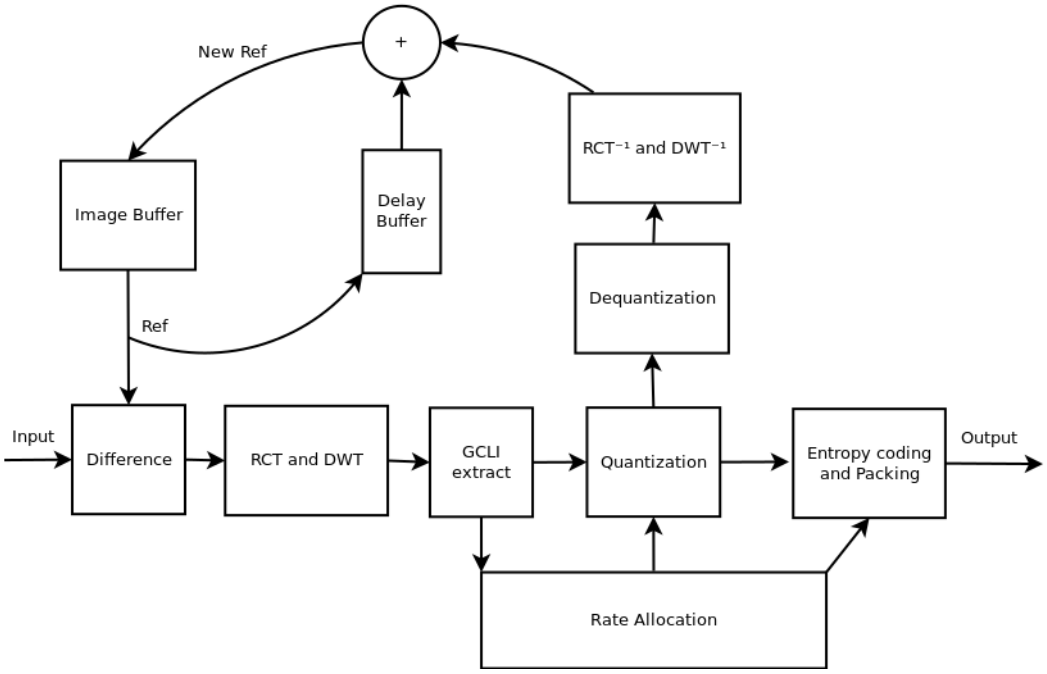


Figure 3.1: Block diagram of the first ITICO coder

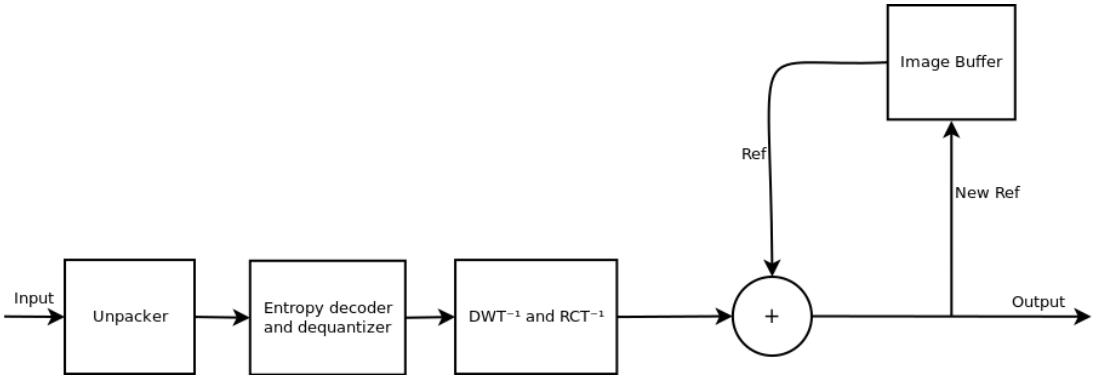


Figure 3.2: Block diagram of the first ITICO decoder

Resources consumption

ITICO complexity

Processing Unit	Memory	Complexity in LUTs
Encoder		
TICO baseline	380kB	15000
Quantization^{-1}	0	1400
Horizontal DWT	Half a line	2000
Vertical DWT	2 lines	1000
RCT^{-1}	0	250
Matrix difference	0	200
Matrix addition	0	200
DDR controller	112kB	12000
Delay buffer	10 lines (225 kB)	100

TOTAL	**767 kB**	**31950**
Decoder		
TICO baseline	180kB	13500
Matrix addition	0	200
DDR controller	112kB	12000
TOTAL	**292 kB**	**25700**

Table 3.1: Complexity and memory increase of ITICO basic concept for video 4K 60fps and hardware frequency at 300MHz for the DWT and 225MHz for the packing blocks

As shown in figure 3.1, seven elements need to be added to the TICO base to form the new ITICO: a matrix difference, a matrix addition, a dequantization, an inverse RCT and DWT, an Image buffer with its controller and a Delay buffer. The matrix addition and the Image buffer are the only elements that increases the resources of the decoder. The resulting increase of complexity can be observed in table 3.2. The amount of LUTs estimated in this chapter is a theoretic approximation. These figures should be checked with a real hardware implementation but this work is out of the scope of this master thesis.

The storage of an entire reference frame requires an important quantity of memory: 53MB. Therefore, an external DDR buffer is required because the internal FPGA memory is sufficient. This DDR buffer needs an important bandwidth for 4K sequences: 4096*2160*60*16*3*2 = 50960.8Mb/s = 6370.1MB/s. A 32-bits DDR3 memory with a bandwidth of 6400 MB/s or a 32-bits DDR4 memory with a bandwidth 9600 MB/s to compensate for the inefficiencies of DDR memory could be used.

In terms of latency, the matrix difference does not require buffering and the dequantization and inverse transforms are performed in parallel with the main algorithm thus do not add anything to the latency.

Results

On figure 3.3, the increase of quality of the frames can be observed. The still sequence chosen is the first image of the Richter sequence(figure A.1) repeated 25 times. The purple horizontal lines are the PSNR[1] result of TICO at different compression rate, the lowest being at 1bpp, the one just above 2bpp, etc. The lowest the bitrate, the biggest the gap of quality between TICO lines. This gap proves that TICO has difficulties in maintaining a good quality on low bitrates for desktop content.

The other curves are the PSNR evolution of this first version of ITICO at several low bitrates: 1bpp, 1.6bpp[2] and 2bpp. The PSNR increases at each iteration, proving that the concept is valid and that desktop content compression is a good use case for ITICO as it greatly improves the quality of the frames within a few iterations thanks to the still nature of desktop content. After 25 frames, the green ITICO curve at 1bpp reaches the same quality than TICO a 7bpp. At higher bitrates, the image after compression gets identical to the input image meaning that the PSNR is infinite and can not be represented on the graph.

[1]PSNR of Imagemagick used, this marker is to be understood as a tool to observe the relative evolution of quality of the compression and not as an absolute marker of quality as the weights used in those results are optimized for visual quality

[2]The compression rate of 1.6 bpp allows a 4096x2160 resolution at 60 fps video sequence to fit into a 1Gb Ethernet connection. Indeed, the bandwidth required is 4096*2160*60*1.6=850Mb/s, leaving space for the audio stream

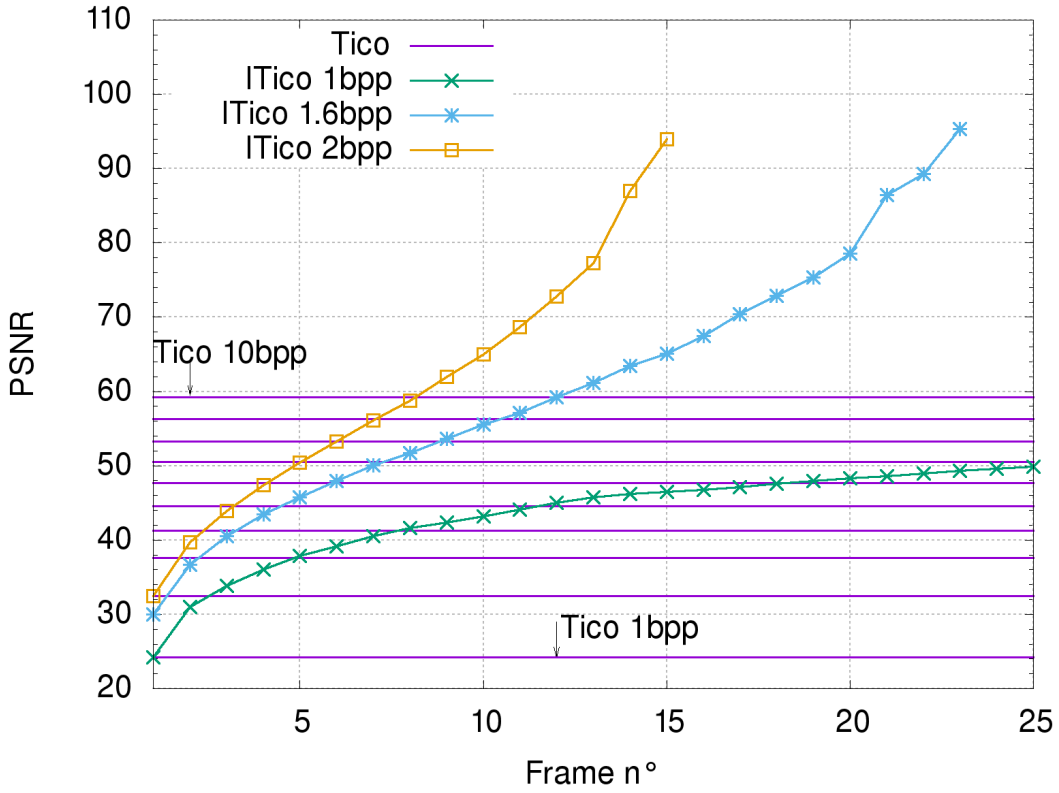
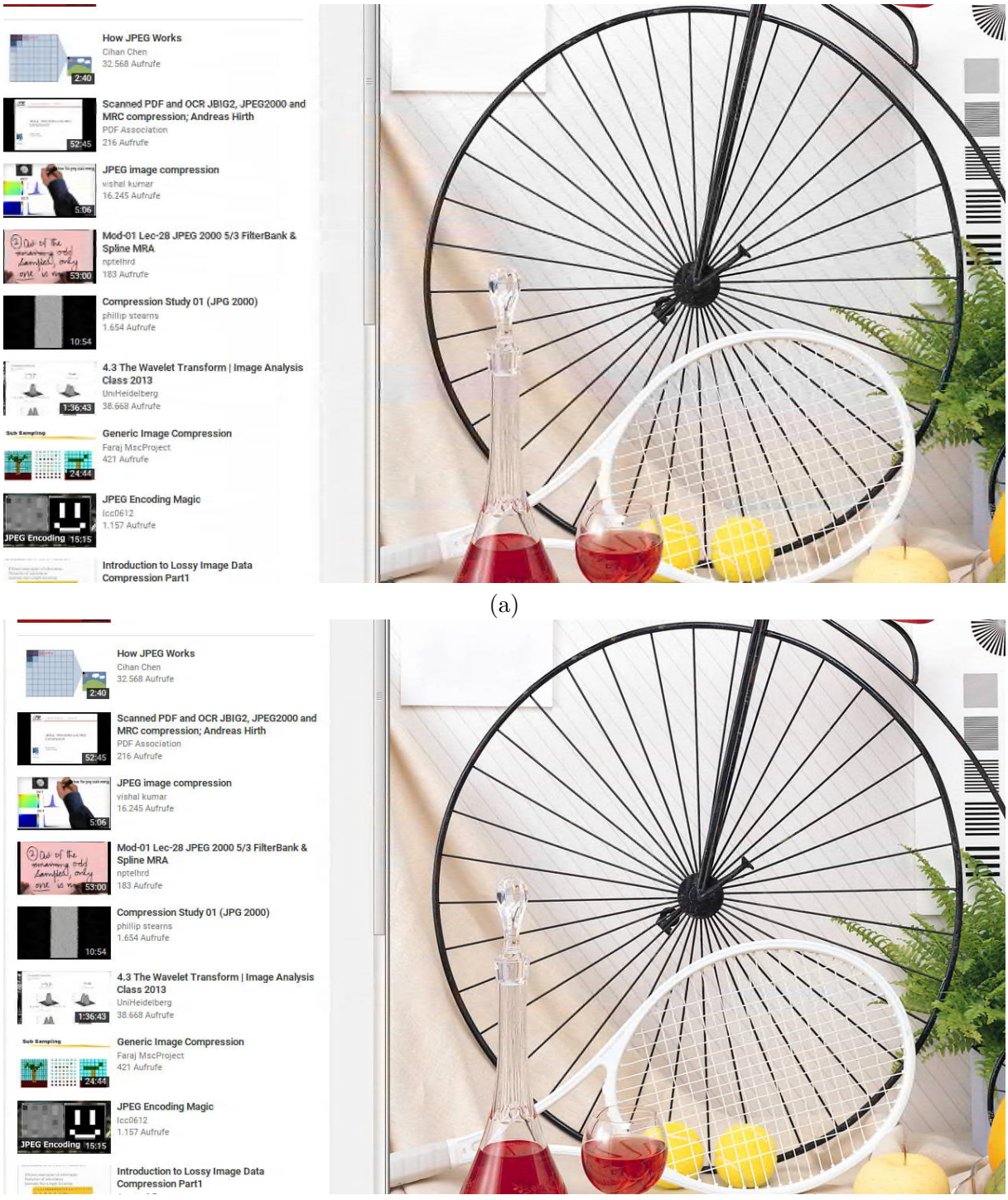


Figure 3.3: First version ITICO results on a sequence of still images

The visual quality improvement can be observed in figure 3.4. The first frame (a) of the still sequence is coded with TICO and important visual degradation can be seen but the second frame (b) is already visually acceptable thanks to the interprediction mechanism.



(a)

(b)

Figure 3.4: Zoom on a part of the still sequence at 1 bpp. (a) first frame, (b) second frame

Conclusion

The simple interprediction mechanism presented here is efficient for video sequences with few or no motion. However, it has several flaws:

- This mechanism is useless and even counter-productive if there is a lot of movement. Indeed, movement creates a problem called "motion artifact" that can be observed in figure 3.5 in the "ghost streets" due to a zoom on the map whereas the face of the lady remains

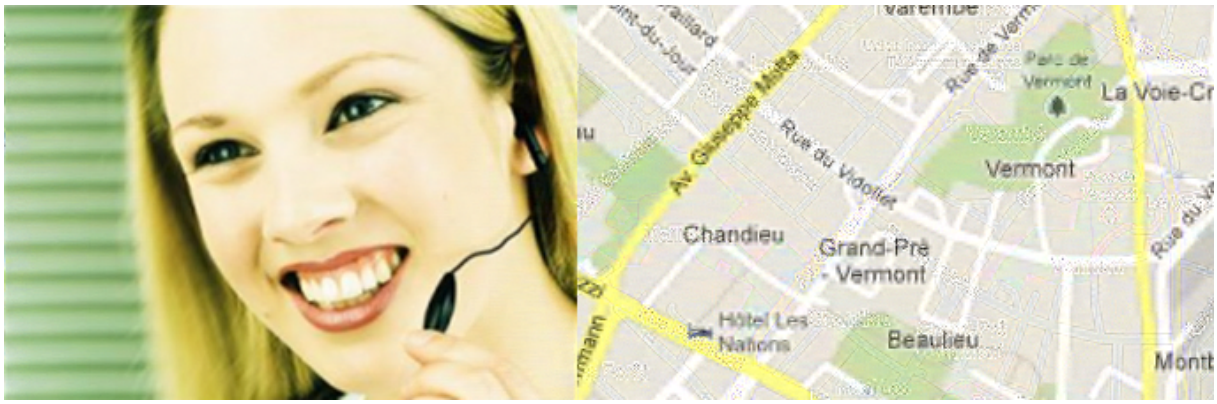


Figure 3.5: Motion artifacts generated by the compression of the first version of ITICO at 1bpp

unmodified as it is not moving.

- In case of error during the transmission of an encoded difference, the decoder is not able to recover from it and the error will remain during the rest of the sequence.
- The memory bandwidth of the frame buffer required to achieve real time compression is not negligible and increases the power consumption and the cost.
- The complexity is greatly increased by the DDR controller and by the inverse transforms at the encoder. In addition to the DDR buffer, the FPGA memory required is also increased.

This is the reason why several mechanisms were developed to mitigate those flaws.

3.2 Difference after DWT

Description

As the DWT and RCT are linear operations (neglecting the rounding induced by the RCT), placing the matrix difference after the transforms is allowed and it simplifies the coder as shown in figure 3.6. However, the decoder remains nearly the same as the inverse transform is still performed to produce the output frame as shown in figure 3.7. This change of place within the codec results in a codec less complex and it can also open the way the new possible improvements as those that will be discussed later.

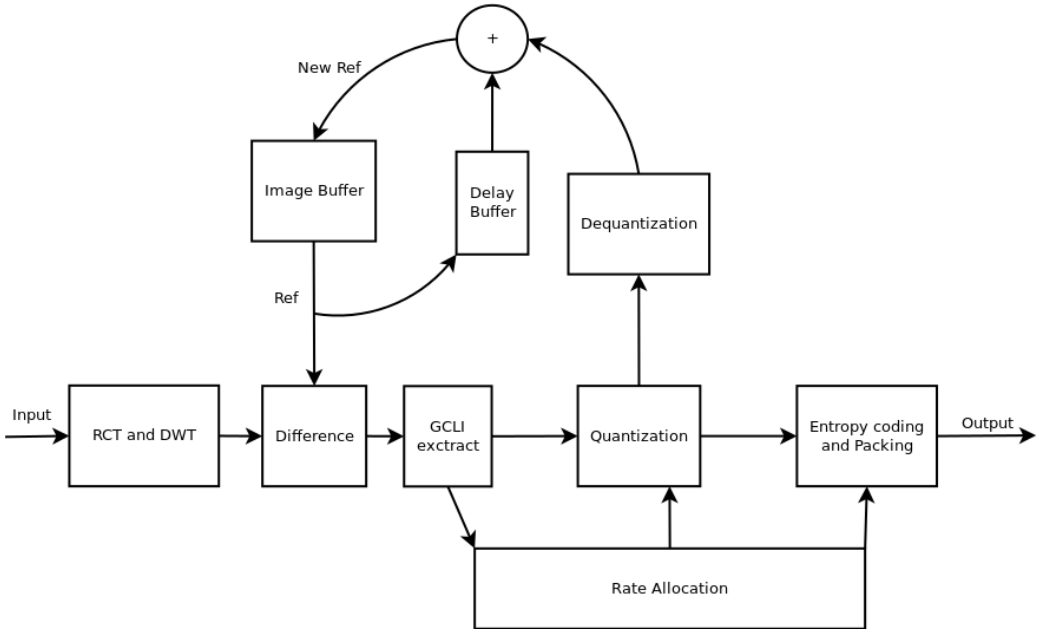


Figure 3.6: Block diagram of ITICO coder with difference after DWT

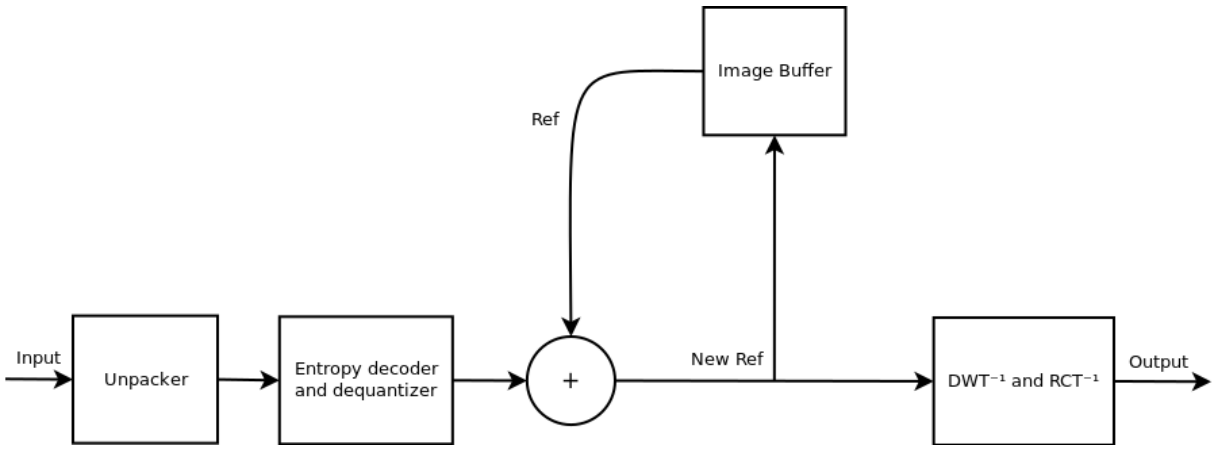


Figure 3.7: Block diagram of ITICO decoder with difference after DWT

Resources consumption

ITICO complexity

Processing Unit	Memory	Complexity in LUTs
Encoder		
TICO baseline	380kB	15000
Quantization^{-1}	0	1400
Matrix difference	0	200
Matrix addition	0	200
DDR controller	112kB	12000
Delay buffer	10 lines (225 kB)	100
TOTAL	**717 kB**	**28700**
Decoder		
TICO baseline	180kB	13500

Matrix addition	0	200
DDR controller	112kB	12000
TOTAL	**292 kB**	**25700**

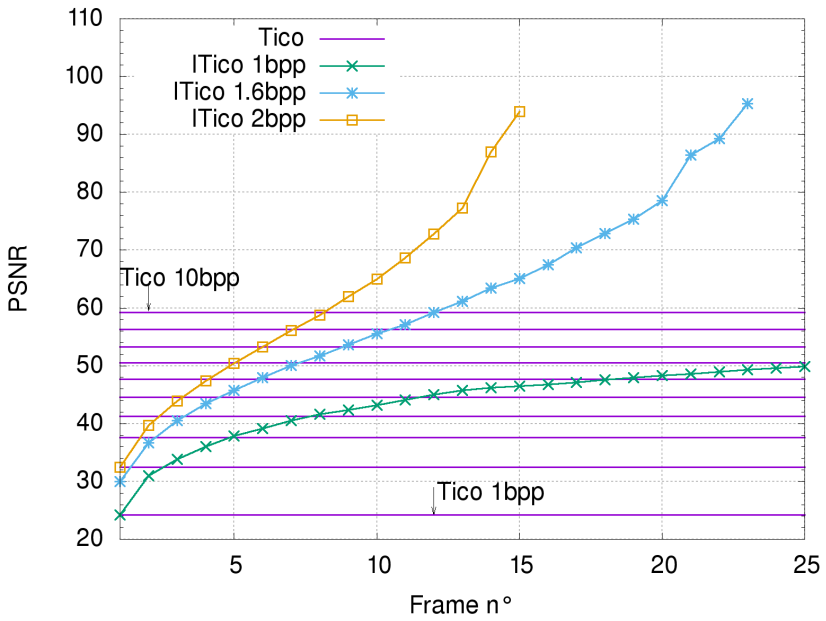
Table 3.2: Complexity and memory increase of ITICO with difference after wavelet for 4K 60fps video and hardware frequency at 300MHz for the DWT and 225MHz for the packing blocks

The simple change of place of the difference allows a big reduction a resources needed for the encoder. From 31950 LUTs, it goes down to 28700. The decoder, however, is not impacted by this change in terms of resources consumption.

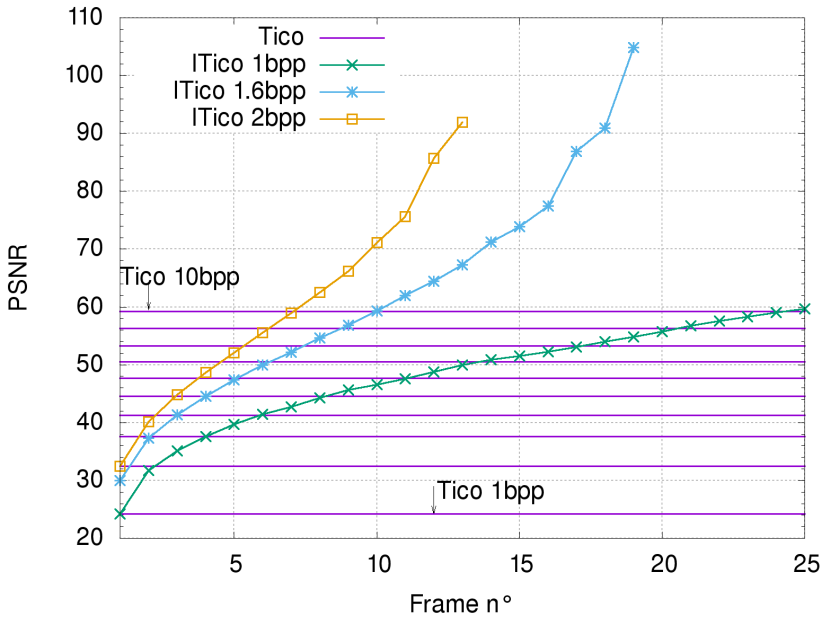
Results

A side effect of moving the difference after the transforms is an improvement of quality because the reference frame is not degraded at each iteration by the rounding induced by the inverse color transform.

It can be observed on figure 3.8 that this effect has a certain impact. The 1bpp curve goes from a quality equivalent to TICO at 7bpp to a quality equivalent to TICO at 10bpp after 25 frames, and the other curves reach a higher quality at a higher speed.



(a)



(b)

Figure 3.8: Comparison of ITICO results with difference placed before (a) and after (b) the DWT on a sequence of still images

Conclusion

The simple fact of changing the place of the interprediction mechanism within the codec allows to lower the resources needed and to improve the results. Furthermore, it opens the way to other improvements such as the Decision process described in the next section.

3.3 Decision mechanism

Within video sequences, some objects move and others remain still. As observed in figure 3.5, the moving parts of the video create motion artifacts if the coefficients related to them are coded with the interprediction scheme of ITICO. Therefore, a process deciding which coefficients in each subband should be coded using a simple TICO scheme is needed. This process is called the Decision mechanism. It allows every still parts of the sequence to get a high visual quality thanks to the intreprediction while the moving parts are simply encoded with the TICO scheme. Furthermore, the coefficients related to moving objects benefit from a higher data budget as still coefficient require only a small budget.

Description

The decision mechanism decides whether a group of eight GCLI's within the same subband and their data should be "intra-coded" (the coefficients remains the original ones) or "inter-coded" (the values of the coefficient are those of the difference between the current frame and the reference frame) as shown in figure 3.9. The criteria for this decision is the comparison of the sum of corresponding groups of GCLI's: if the sum of the group of GCLI's belonging to the difference is lower than the sum of the original GCLI's then the difference is advantageous and consequently the original data is replaced by the difference. A bit flag is set to indicate to the decoder that this group is inter-coded.

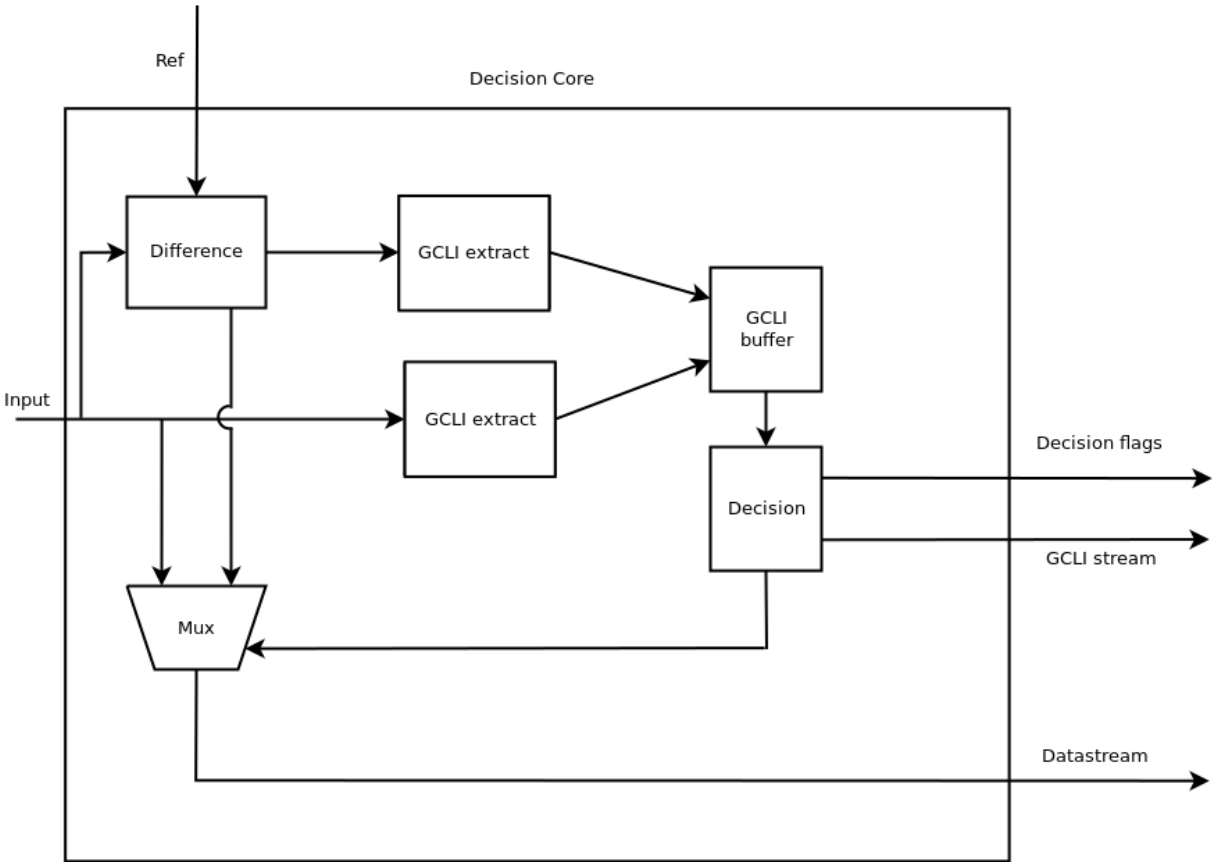


Figure 3.9: Block diagram of the decision mechanism

The choice of grouping GCLI's by eight is made to work in symbiosis with the filtering of irrelevant GCLI's of TICO. Indeed, if a group of 8 GCLI's is inter-coded, it means that their value will be lower and thus have more probability to become irrelevant and be filtered.

Grouping the GCLI's by eight is also considered a good trade-off between quality improvement during motion and the degradation caused by the overhead reducing the budget for raw data. Thanks to this decision, the motion artifacts are removed as moving objects are intra-coded while still objects in the video sequence remains with a high quality as they are inter-coded.

In addition to the overhead bit per group, a single bit flag in the picture header is set to indicate if the image is encoded in a simple TICO or in ITICO mode. If the image is encoded in TICO mode, then there is no overhead due to the decision flags. It allows to send the first image in TICO mode and thus avoid a loss of quality. It is especially important at low bitrates (1-2 bpp), which is the goal of ITICO. It also allows a certain kind of compatibility with previous versions of TICO. Indeed, header bits were reserved for future improvements and were set to 0 and thus an ITICO decoder can easily decode a previous TICO codestream.

Resources consumption

ITICO with Decision complexity

Processing Unit	Memory	Complexity in LUTs
Encoder		
ITICO Diff after DWT	717 kB	28700
GCLI extract and buffer	16*4 bits	200
Decision	0	700
Decision flags packing	0	500
TOTAL	**717 kB**	**30100**
Decoder		
ITICO Diff after DWT	292 kB MB	25700
Decision flags unpacking	0	500
TOTAL	**292 kB**	**26200**

Table 3.3: Complexity and memory increase of ITICO with decision for 4K 60fps video and hardware frequency at 300MHz for the DWT and 225MHz for the packing blocks

The increase of resources at the encoder and decoder shown in table 3.3 is reasonable and the cost is worth it as the results below reveal.

Results

The effects of the decision mechanism can be observed in figure 3.10[3], during the first still subsequence (frame 1 to frame 61), the overhead of the decision flags is causing a lower quality. However, when the motion begins (a zoom on the map), the decision allows not to drop under the TICO curve. Furthermore, as the codec with decision must not recover from all the motion artifacts generated during this motion, it performs better on the following still and moving (zooms) sequences.

This decision mechanism also allows ITICO not to degrade the quality of natural content that has the particularity to have more motion and to be noisy. Indeed, as shown in figure 3.11, ITICO without the decision mechanism degrades the quality of the compression whereas ITICO with the decision mechanism reach globally the same results as TICO. The natural content used

[3]The maps sequence is composed of goolge maps that sometimes moves (zooms and translations) while the other windows remain still.

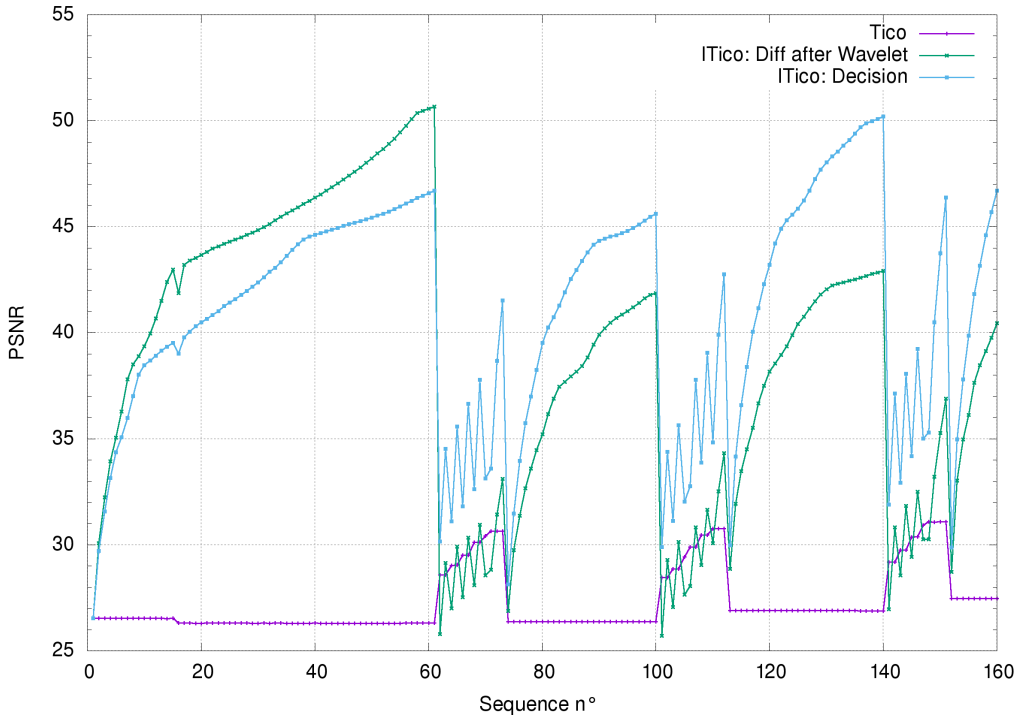


Figure 3.10: Comparison of ITICO with and without the decision mechanism on the first 160 frames of the maps sequence (A.2) at 1bpp

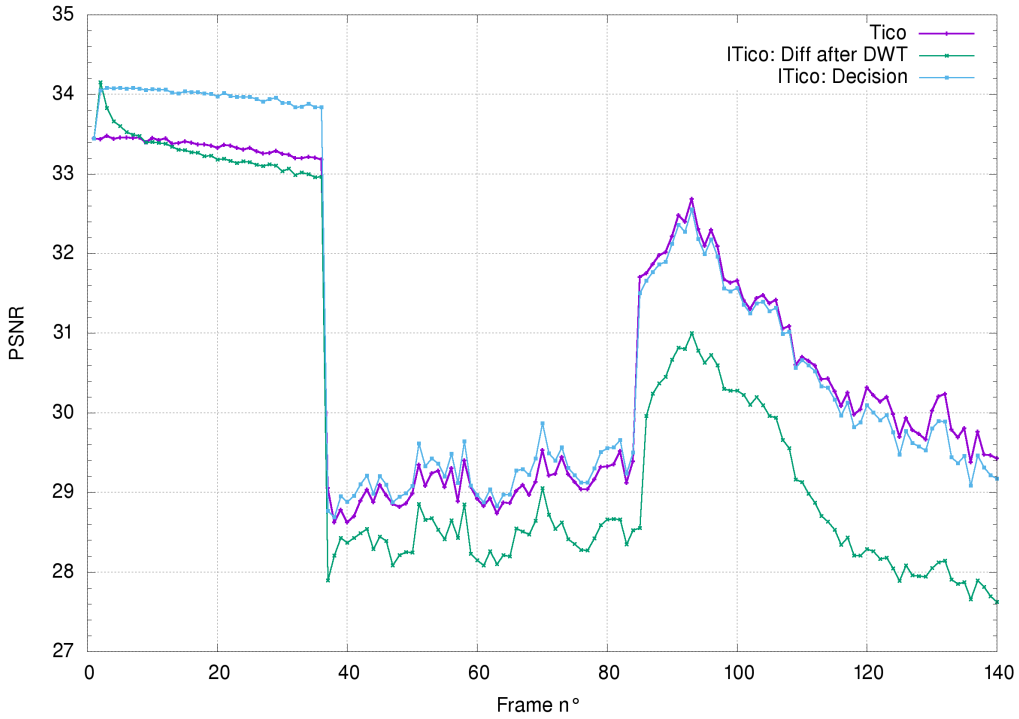


Figure 3.11: Comparison of ITICO with and without the decision mechanism on a sequence composed of the pendulus sequence (A.6), the crowdrun sequence (A.4) and the parkjoy sequence (A.5) at 1bpp

for this comparison is made of three different sequences to simulate a change of scene during a movie. During thoses changes, the ITICO without decision create some motion artifacts.

Conclusion

This decision process allows to completely remove the problem of motion artifacts at a very low cost. Removing those motion artifacts allows to reach better overall results in desktop sequences but it also allows havet the same quality as TICO for natural content which is good as TICO is already efficient enough for natural content. It is therefore an indispensable feature of the codec as there is almost always some degree of motion within video sequences.

3.4 Refresh

The goal of the refresh mechanism is to mitigate the problem of transmission errors or the unexpected change of input source at the decoder. Indeed, if one of those two event appears, the reference frame of the decoder does not match the reference frame of the correspond coder, leading to an unwanted drift between the coder and the decoder. The only way to remove this drift is to send intra-coded data and, without a refresh mechanism, there is no obligation to send intra-coded data.

The refresh mechanism developed for ITICO is linked to the decision mechanism as it overrides the natural decision to force some groups of GCLI's to be intra-coded.

3.4.1 Simple refresh

Description

The idea is to progressively force each group of GCLI's to be intra-coded within a defined frame interval called the refresh rate no matter what the decision mechanism decides. By default, it is a 20 frames interval but it can be increased or lowered. The mathematical formula used for the refresh is:

$$(FrameNumber + GroupNumber) \,\%\, RefreshRate$$

The % is a modulo operator. If this formula equals 0 then the group is intra-coded. The purpose of the FrameNumber term is to change the groups refreshed at each frame. The main reason a progressive refresh is used is to avoid a flickering problem caused by the sudden degradation of the image quality if a simple I frame was used as refresh mechanism. Indeed, ITICO is characterized by its low latency thus it can not allocate more budget to a specific frame and reduce the budget of others to compensate in the same way hybrid codecs do for their I frames. However, ITICO can allocate more budget to some parts within a frame thanks to its rate allocation and therefore a progressive refresh is the best choice.

Resources consumption

The refresh is an element using a little more resources at the decision block on figure 3.9. The estimated cost of this refresh is 200 LUTS thus the ITCO coder is now at 30300 LUTs.

Results

The refresh mechanism allows to be sure that the drift caused by a transmission error or any unexpected event fades away within the refresh rate interval as shown in figure 3.12 where a sudden change of input channel is simulated at the decoder. The original basket sequence (A.3)

is changed by the maps sequence (A.2) after the frame 60 causing the reference frame of the decoder to be invalid. The green curve is the version of ITICO without any refresh mechanism but with the decision mechanism. The blue curve is the result of this refresh mechanism and the orange curve is the result of the multi-resolution mechanism discussed in the next paragraph. The versions of ITICO with the refresh mechanism can recover from this unexpected change of input. However, the refresh mechanisms have the side effect to lower the quality of the compression as observed in the 60 first frames of figure 3.12 but it is an acceptable price to pay to ensure error resiliency.

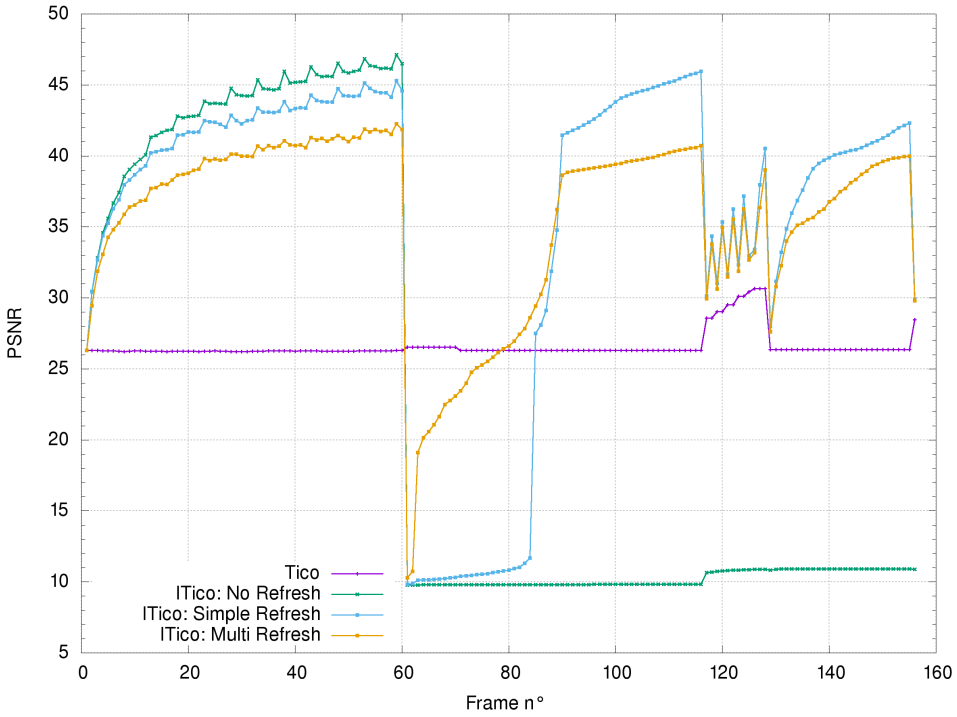


Figure 3.12: Comparison of the different refresh mechanisms of ITICO during a sudden change of input source at 1bpp

3.4.2 Multi-resolution refresh

In figure 3.12, it can be observed that the blue curve recovers an acceptable quality only at the end of the 30 frame interval required. This is caused by the fact that, by lack of chance, the groups within the lowest subbands (thus bearing the more information) are the last refreshed. The frames during this interval are thus not watchable.

Description

Making sure that the lowest subbands are refreshed quickly enough allows to avoid this problem. This is the reason why the multi-resolution refresh is created. The idea is to adapt the refresh rate to the size of each subband with an upper bound at a maximum refresh rate (30 by default). As the lower frequencies are smaller they are refreshed more often. Therefore, in case of errors, the lowest resolution is the first to be fully recovered, followed by the second lowest, etc. This is why it is called a multi-resolution refresh. The formula used for this new refresh mechanism is:

$$(FrameNumber + GroupNumber) \,\%\, min(SubbandGCLISize * 3,\ RefreshRate)$$

Resources consumption

This multi-resolution refresh is a little more complex thus its estimated cost is 300 LUTs. The coder is now at 30400 LUTs.

Results

In figure 3.12, it can be observed that both version of the refresh mechanism need the 30 frame interval to recover but the multi-resolution version recovers an acceptable quality earlier, making the sequence watchable even though the recovering process is not entirely finished. As expected, a more constraining refresh mechanism slows down even more the quality increase of a still sequence. In figure 3.13, the results of the different refresh mechanisms in comparison with the simple difference after wavelet version at a compression of 1bpp can be observed. Clearly the mutli-resolution refresh is the lowest but the quality of the image is still very quickly visually acceptable. Furthermore, the refresh rate could be increased to reach better results if needed.

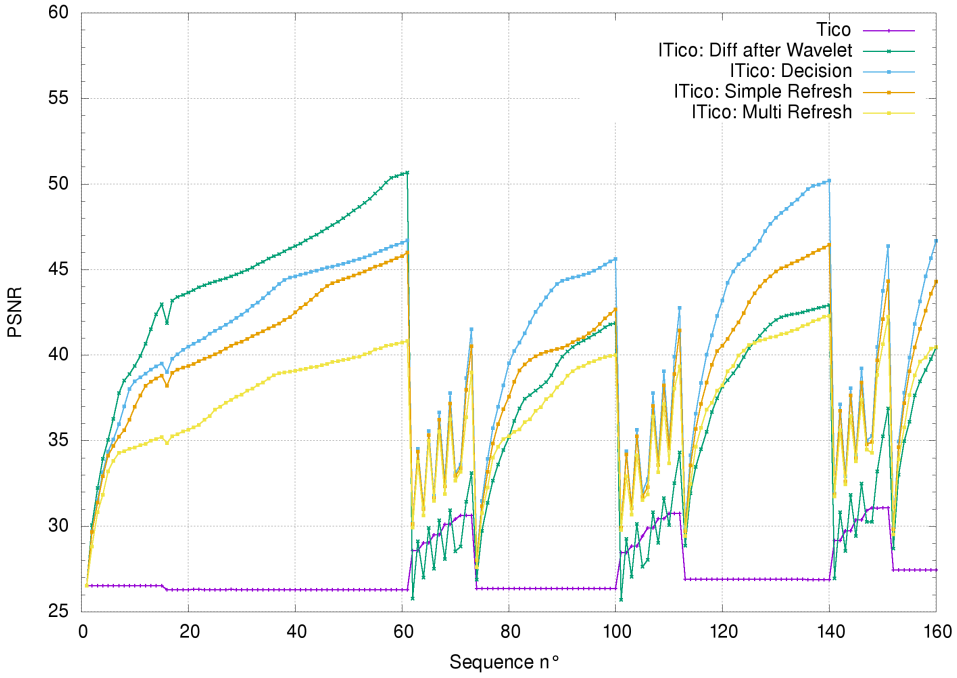


Figure 3.13: Comparison of the different version of the refresh mechanism and without any refresh mechanism on maps sequence at 1bpp

Conclusion

The refresh mechanism is a trade-off between quality of the compression and error resiliency. In many use cases, transmission errors and unexpected change of input source at the decoder are unavoidable and consequently it is imperative to recover from those events. A progressive refresh mechanism seems the best way to mitigate the impact of such events although it is true that a few frames are "lost" during the recovery process. Using the multi-scale representation of the wavelet domain allows to minimize the visual impact of those "lost" frames at a relatively small cost in quality .

If there is a valid hypothesis that no unexpected event can alter the datastream between the coder and the decoder then the refresh mechanism should be removed to obtain a better

compression quality.

3.5 Reference frame compression

Memory access can be a bottleneck in hardware implementations of a codec. Indeed, for a 4K video at 60 fps, a memory bandwidth of 4096*2160*60*16*3*2 = 50960.8Mb/s = 6370.1MB/s is required for the reference frame buffer; the factor 2 comes from the fact that reading and writing are needed at the same time. The 16*3 represents the depth of the coefficients multiplied by the number of components. Such a bandwidth requires an expensive 32-bits bus DDR4 memory to use as the buffer. Compressing this reference frame when it is stored in the buffer and decompressing it when it is accessed for the difference as shown in figure 3.14 would allow to reduce this bandwidth.

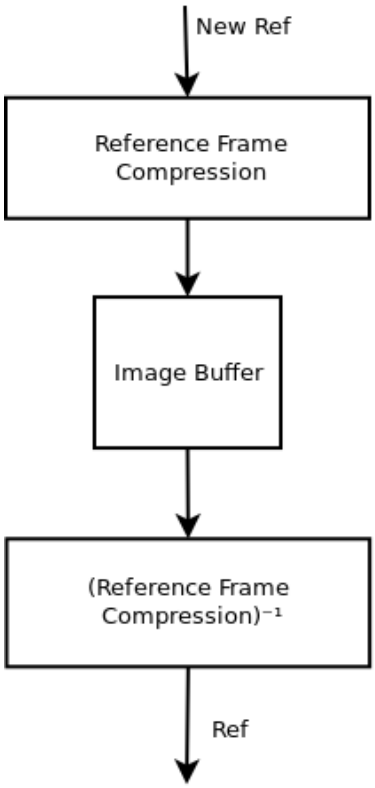


Figure 3.14: Reference frame compression

3.5.1 TICO for reference frame compression

Description

The first Reference Frame Compression (RFC) system used is the version 3 TICO with the only particularities that the weights defined for the main algorithm are reused and that no transform is performed thanks to the fact that the difference is placed after the wavelet transform. Indeed, if the difference was placed before the DWT, an inverse transform would be required when the reference frame is decompressed.

Compressing the reference frame puts an upper bond to the quality of the image but it does not have an impact on the speed of the quality increase. The upper bound can be estimated. Let's call Q_b the quality of an image compressed with TICO at a bitrate b and Q the maximum quality reachable by ITICO with reference frame compression, B1 the bitrate chosen for the

reference frame and B2 the bitrate chosen for the main compression. The output frame of the decoder is the addition of the reference frame coded at bitrate B1 and the input compressed difference at bitrate B2. Therefore, the upper bound in quality of the output is between:

$$Q_{B1} < Q < Q_{B1+B2}$$

Resources consumption

ITICO with RFC complexity

Processing Unit	Memory	Complexity in LUTs
Encoder		
ITICO Diff. after DWT	717-74= 643 kB	28700-8000= 20700[4]
TICO coder without transforms	380 kB	10125
TICO decoder without transforms	180 kB	8625
TOTAL	**1203 kB**	**39450**
Decoder		
ITICO Diff after DWT	292-74=218 kB	25700-8000= 17700
TICO coder without transforms	380 kB	10125
TICO decoder without transforms	180 kB	8625
TOTAL	**778 kB**	**36450**

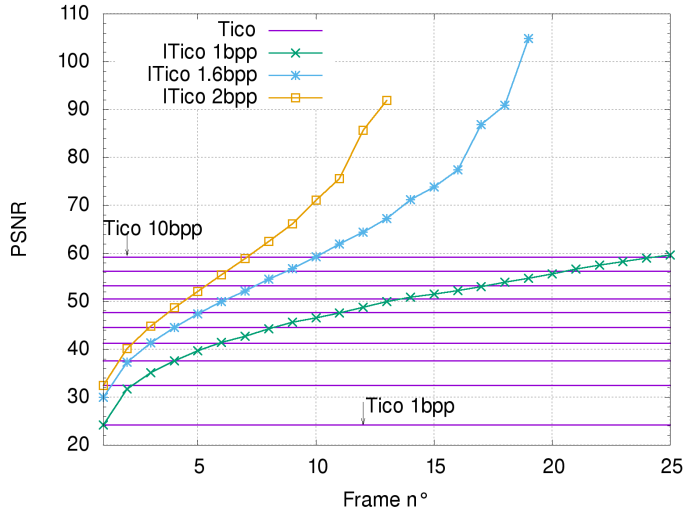
Table 3.4: Complexity and memory increase of ITICO with TICO for reference frame compression for 4K 60fps video and hardware frequency at 300MHz for the DWT and 225MHz for the packing blocks

As shown in table 3.4, the complexity added by using TICO for the reference frame compression is important but it allows to use a 8-bit DDR4 or DDR3 buffer and a corresponding controller less complex. Indeed, the bitrate chose for the RFC is 6bpp thus the bandwidth requires is 796.2 MB/s. The bandwidth of a 8-bit DDR4 is 2400M B/s and the bandwidth of 8-bit DDR3 is 1600 MB/s.

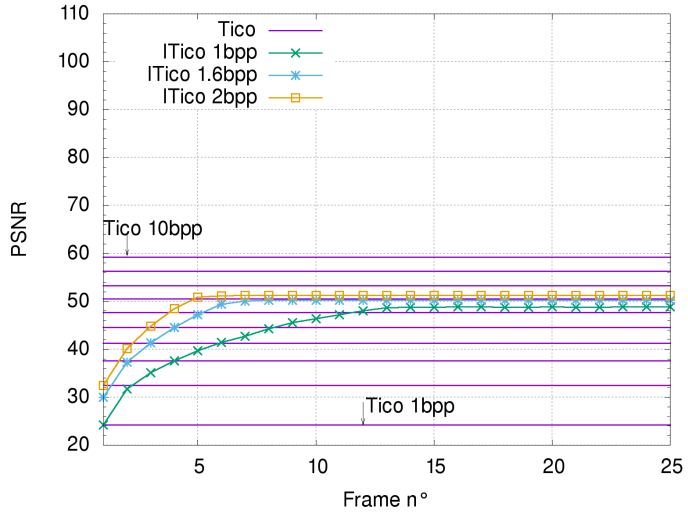
Results

By experience, the bitrate chosen for the reference frame compression is set at 6bpp because it gives a visually acceptable upper bound in quality. The results of this version of ITICO are shown in (b) of figure 3.15. The upper bound is above TICO at 6bpp, as estimated and the speed of the quality increase is not affected by the reference frame compression before it reaches the bound.

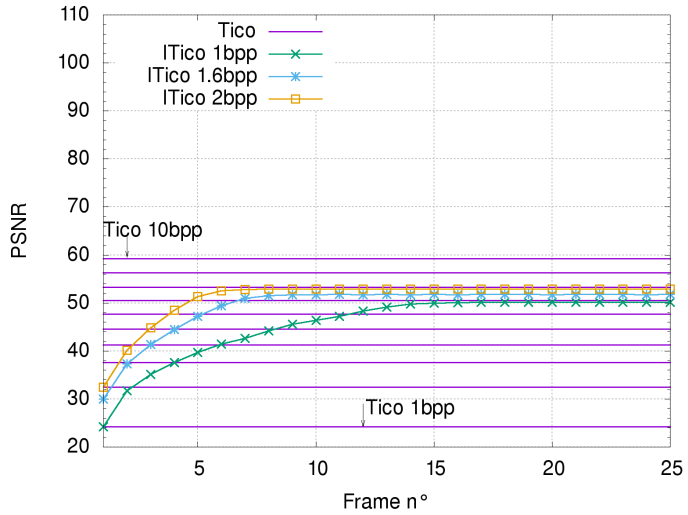
[4]RFC allows to use 8-bit bus DDR4 which requires a controller less complex



(a)



(b)



(c)

Figure 3.15: Results of ITICO on a sequence of still images; (a) Difference after wavelet; (b) Reference frame compression at 6bpp; (c) Simplified reference frame compression at 9bpp

3.5.2 Simplified TICO for reference frame compression

Description

The complexity increase of using TICO for the reference frame compression is too important. Therefore, a solution to lower the required resources is to simplify the TICO used for this reference frame compression. The simplifications made are:

- No filtering of the sign bits nor of the irrelevant GCLI's.
- No rate allocation window, thus each precinct is quantized to reach the average precinct budget.
- Uniform Deadzone quantification instead of Center-rounding quantification. This quantization is thus naturaly done by the packing for the coder and is much simpler for the decoder.
- No entropy coding on the GCLI's.

Resources consumption

Simplified TICO complexity

Processing Unit	Memory	Complexity in LUTs
Encoder		
Data buffer	3 lines	100
Rate allocation	0	1200
GCLI extract and packing	0	700
Data Packing	0	1200
TOTAL	**75kB**	3200*1.5= **4800**[5]
Decoder		
Input Mux	0	200
Data buffer	3 compressed lines	100
GCLI buffer	3 lines of RAW GCLI	100
GCLI unpacking	0	300
Data unpacking	0	1200
Quantization^{-1}	0	500
TOTAL	**25kB**	2400*1.5= **3600**

Table 3.5: Complexity and memory of each processing unit in simplified TICO for video 4K 60fps and hardware frequency at 300MHz for the DWT and 225MHz for the packing blocks

Thanks to those simplifications, the complexity of the reference frame compression is lowered as shown in 3.5. The estimated complexity of the ITICO coder and the decoder with this simplified reference frame compression is respectively $28700 - 8000 + 4800 + 3600 = 29100$ and $25700 - 8000 + 4800 + 3600 = 26100$ LUTs and the FPGA memory needed is: $717 + 75 + 25 = 817$ and $292 + 75 + 25 = 392$ kB. The bitrate chosen for this RFC is 9bpp thus a memory bandwidth of 4096*2160*60*9*2 = 9555.14 Mb/s = 1194.39 MB/s is required. A 8-bits DDR4/DDR3 is still enough.

It is worth noticing that a trade-off between memory and complexity is possible. Instead of placing the Delay Buffer after the Simplified TICO decoder (figure 3.16), it could be placed

[5]Multiplication factor accounts for control operations, integration, etc.

before the Simplified TICO decoder. It would require less memory (46kB instead of 225kB) but a second Simplified TICO decoder should be added to handle the workload. Another possibility is to remove the Delay Buffer and let the DDR memory handle the workload of giving the required precinct a second time for the matrix addition. In that case, a memory bandwidth of 4096*2160*60*9*3= 14332 Mb/s = 1791.5 MB/s is required thus either a 16-bits DDR3 or the 8-bits DDR4 and a second Simplified TICO decoder are required.

The operations of the reference frame compression are performed in parallel with the main algorithm thus it should not impact the latency.

Results

As TICO is simplified for the reference frame compression, the quality of the reference compression is lowered. To compensate for this degradation of quality, a higher bitrate must be set for the reference frame compression. Through experience, a bitrate of 9bpp for the simplified TICO RFC is considered to give the same results as a bitrate of 6 bpp for the TICO RFC. In figure 3.15, the 9bpp simplified RFC (c) is even slightly superior in quality.

3.5.3 Conclusion

The goal of reference frame compression is to reduce the memory bandwidth required. This improvement allows to use a cheaper DDR memory for the buffer of the reference frame like a 8-bits DDR4 or a 8-bits DDR3.

RFC is a trade-off between quality of the compression of the codec and the cost of its hardware implementation. If it is estimated acceptable to use a higher quality and a more expensive DDR memory, then this feature should not be kept. Naturally, memory bandwidth is not a problem in software implementation but RFC must be simulated to ensure the compatibility between hardware and software implementations of ITICO.

3.6 Final ITICO

Description

The combination of the mechanisms explained previously allows to deal with the flaws identified in the first concept with an acceptable trade-off with the drawbacks they induce. Combined together, they form a codec allowing an important improvement of quality in still sequences although upper bounded, removing the problem of motion artifacts during the moving sequences, requiring a relatively low bandwidth for the memory buffer and allowing to recover from transmission errors. The resulting codec is shown in figures 3.16 and 3.17.

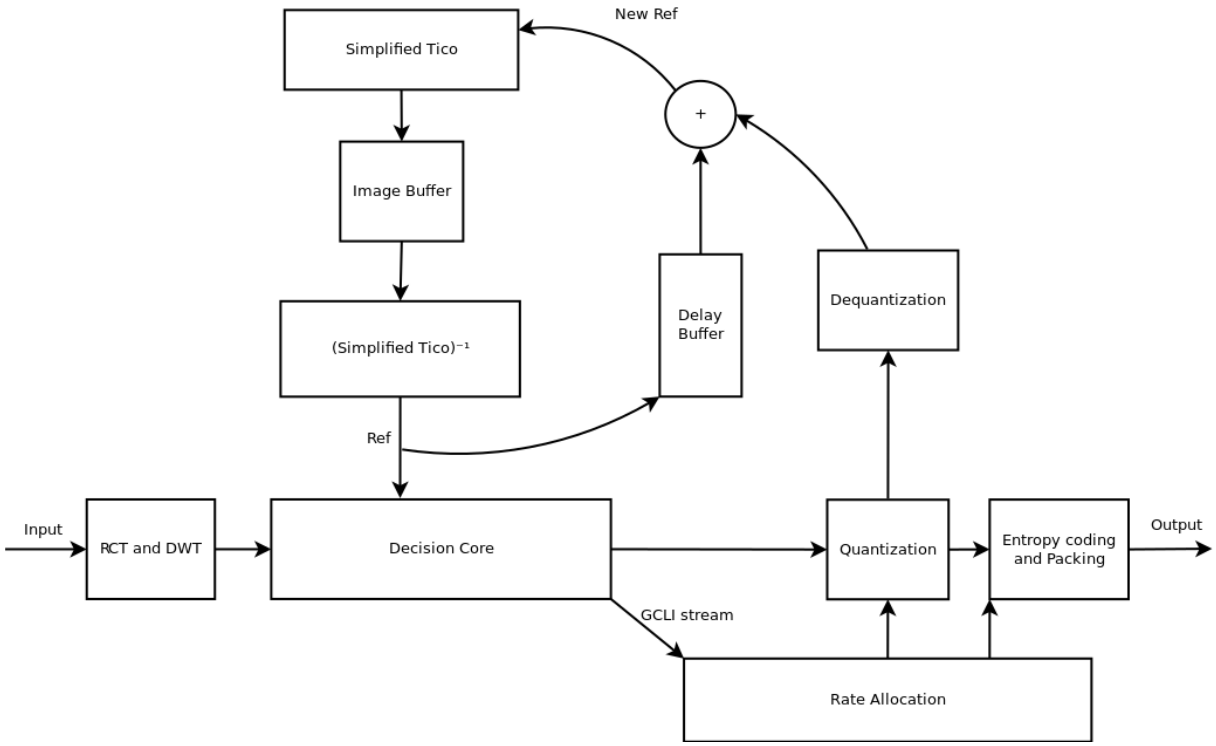


Figure 3.16: Block diagram of the last version ITICO coder

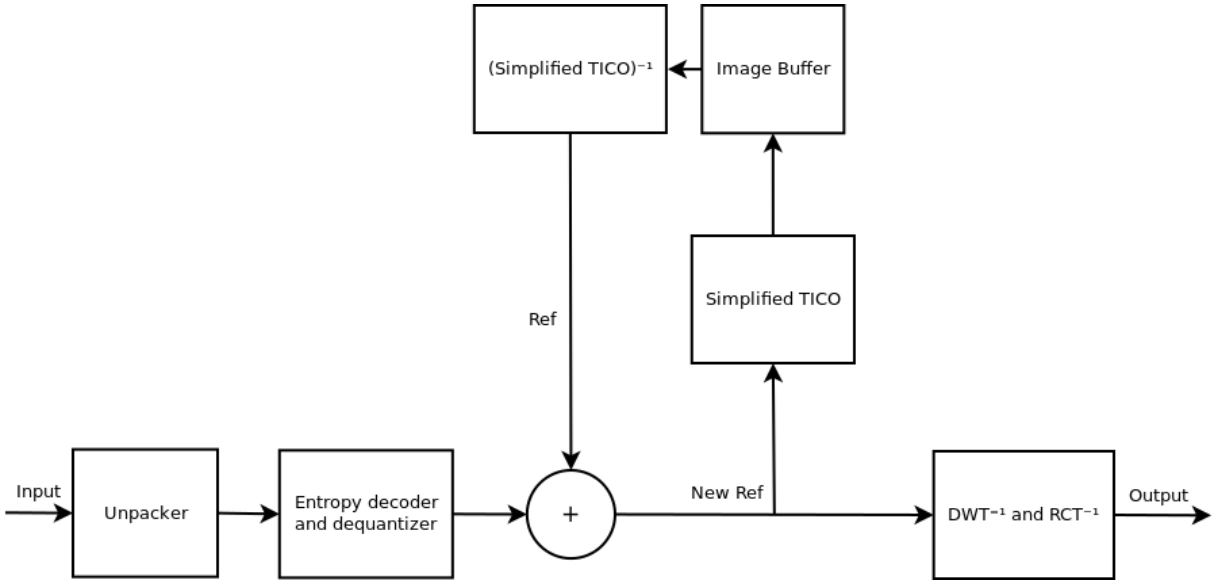


Figure 3.17: Block diagram of the last version ITICO decoder

Resources consumption

Final ITICO complexity

Processing Unit	Memory	Complexity in LUTs
Encoder		
TICO baseline	380kB	15000
Quantization^{-1}	0	1400
Matrix difference	0	200

Matrix addition	0	200
DDR controller	38 kB	4000
Delay buffer	10 lines	100
GCLI extract and buffer	16*4 bits	200
Decision and Refresh	0	1000
Decision flags packing	0	500
Simplified TICO coder	75kB	4800
Simplified TICO decoder	25kB	3600
TOTAL	**750 kB**	**31000**
Decoder		
TICO baseline	180kB	13500
Matrix addition	0	200
DDR controller	38 kB	4000
Decision flags unpacking	0	500
Simplified TICO coder	75kB	4800
Simplified TICO decoder	25kB	3600
TOTAL	**320 kB**	**26600**

Table 3.6: Complexity and memory increase of ITICO with difference after wavelet for 4K 60fps video and hardware frequency at 300MHz for the DWT and 225MHz for the packing blocks

The hardware complexity of the final version of ITICO is shown in table 3.6.[6]. Those amounts are around the double of LUTs in comparison with the TICO coder and decoder but it is still a low complexity, especially for a codec using interprediction. Indeed, a FPGA implementation of a JPEG2000 coder requires 45000 LUTs to handle HD streams and it does not use interprediction [17].

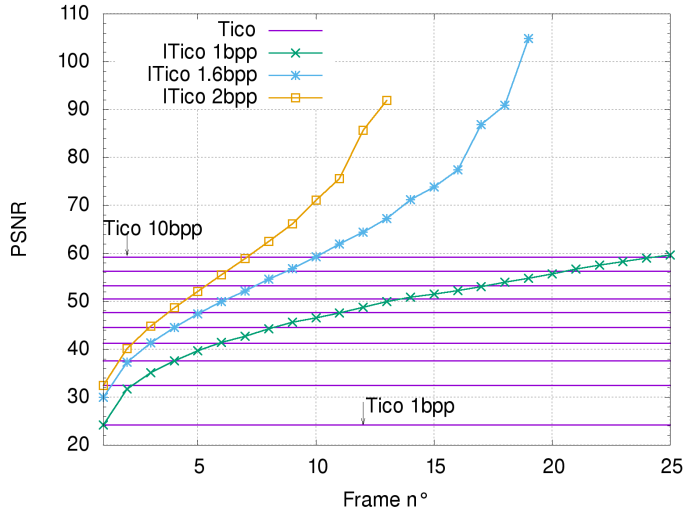
The hardware latency of the codec should remain nearly identical with at the most one more line of latency in the coder cause dy the decision mechanism.

The software latency remains the same as for TICO: one frame. The execution time remains relatively low as the reference frame compression can be computed in parallel with the main algorithm.

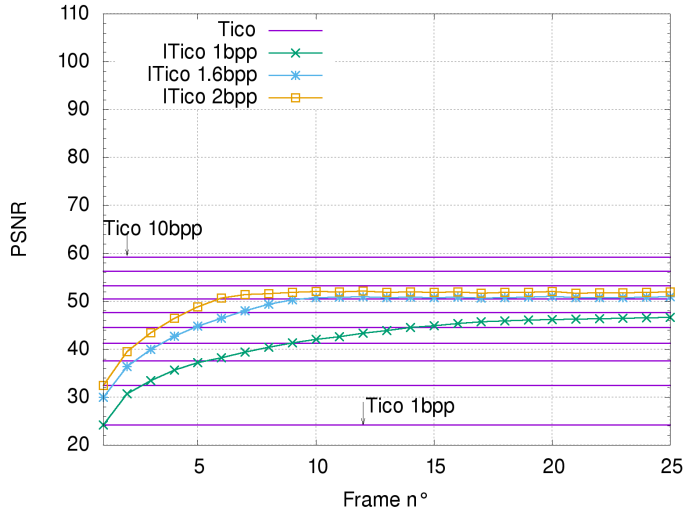
Results

The comparison of the results of this final codec with the difference after wavelet version in figure 3.18 illustrates their differences. The last version is bounded and its quality increase rate is lower than the first concept but it is quite acceptable. Indeed, at 1bpp the last ITICO gets a quality nearly equivalent to TICO at 6bpp in 20-25 frames (half a second at 60fps, a second at 25 fps). At 1.6bpp and 2bpp the quality reached is the quality of TICO at 7bpp in less than 10 frames. It is worth noticing that the second frame achieves a huge improvement of quality compared to the first frame. Indeed, at 1 bpp, the second frame is almost at the quality of TICO at 2 bpp. The same goes for the two other curves. Due to the framerate of normal video sequences, the human eye does not notice the low quality of the starting frame.

[6]Those numbers are just an estimation and would required checking by hardware implementation



(a)



(b)

Figure 3.18: Results of ITICO on a sequence of still images. (a)ITICO with difference after DWT (b) final version of ITICO

In 3.19, the behaviour of the first and final version of ITICO can be observed at 1bpp during motion. At 1bpp the upper bound set by the reference frame compression is not reached because of the refresh mechanism. During the 60 first frames, the last version of ITICO has a lower quality than the simple difference mechanism but the roles are reversed after the 60^{th} frame because of the motion. The last version is better equiped to deal with it and remains at a higher quality than TICO and the simple ITICO during the motion and even during the following still sequences.

ITICO is conceived for a specific use case, desktop content, but it is important to discuss another major use case in compression: natural content. As figure 3.20 reveals, the final ITICO does not lower the quality of the compression thanks to its decision mechanism whereas the first concept does due to the high motion and noisy nature of this content.

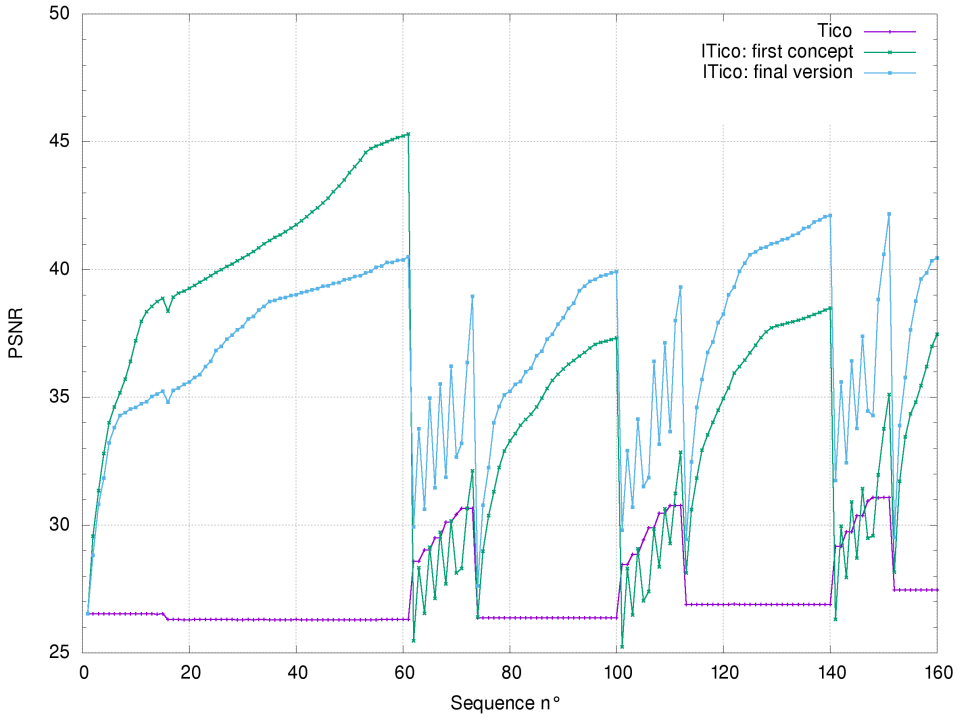


Figure 3.19: Comparison of the first concept and the final version of ITICO on the maps sequence at 1bpp

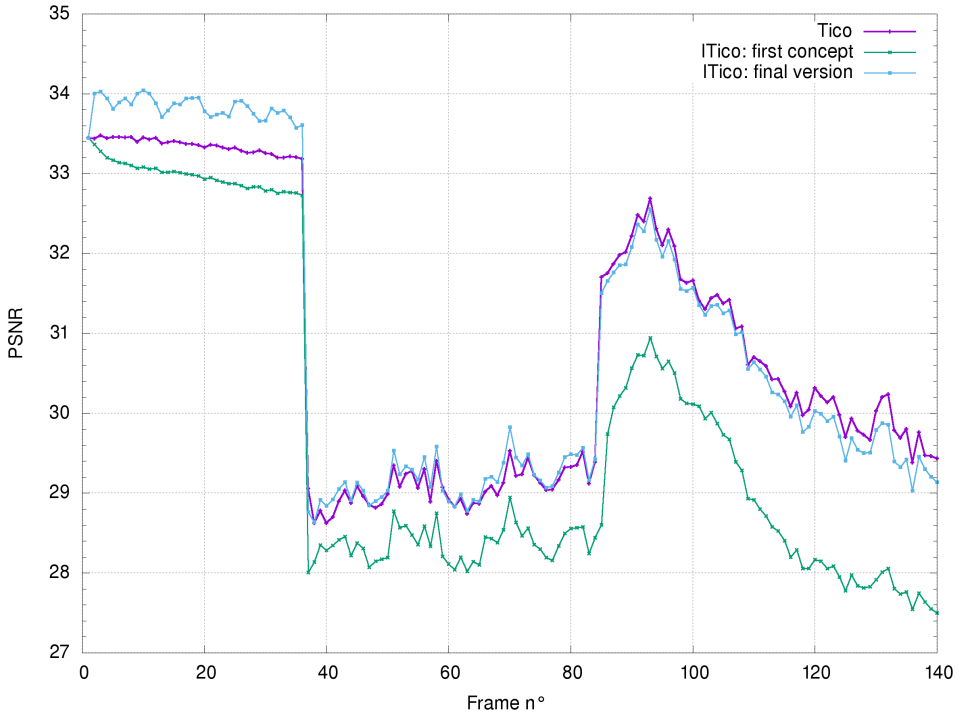


Figure 3.20: Comparison of the first and final version of ITICO on a sequence composed of the pendulus sequence (A.6), the crowdrun sequence (A.4) and the parkjoy sequence (A.5) at 1bpp

Table 3.7 confirms that ITICO is efficient for desktop content but does not have a significant impact on natural and computer generated content.

PSNR results comparison

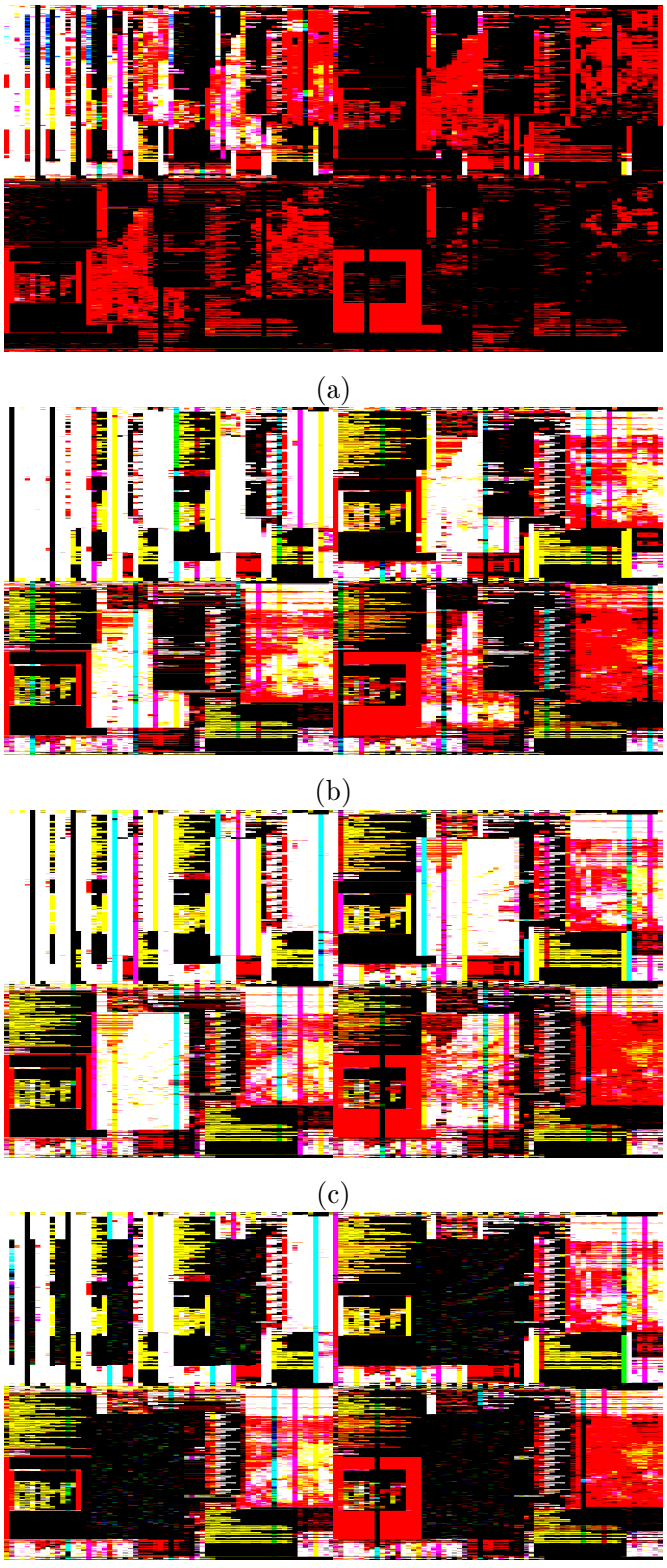
Sequence name	TICO mean PSNR		ITICO mean PSNR	
	1bpp	1.6bpp	1bpp	1.6bpp
Desktop content				
Maps	29.10	33.14	42.86	49.86
Richter	24.90	30.06	48.90	52.56
Basket	27.63	31.60	44.79	48.87
Natural and computer generated content				
Pendulus, Crowdrun and Parkjoy combinaison	30.85	32.54	30.93	32.63
Lake2 and Public University combinaison	30.65	32.94	30.75	33.09
Sintel	45.73	49.21	46.49	50.03

Table 3.7: Comparison of compression results between TICO and ITICO

Another interesting observation is the evolution of the coefficients that are intra or inter-coded. The evolution is shown in figure 3.21. The image (a) is the second frame of the Richter sequence, just after the entirely intra-coded first frame. The low frequencies are highly inter-coded as attested by the strong presence of the white color (the three components are inter-coded) in the top left of the image, and this is a behaviour observable on every sequence, even in high motion sequences.

The high frequencies get gradually inter-coded: the (a) frame only gets some Y component of the high frequencies inter-encoded, the (b) frame which is the sixth frame of the sequence gets a lot more of coefficients and components inter-coded. The (c) frame is located just before the movement of a window and (d) takes place during this movement. The coefficients related to this moving window can clearly be observed as they suddenly get intra-coded.

Vertical bands in black or without one of the color component can be observed. It is the result of the refresh mechanism.



(a)

(b)

(c)

(d)

Figure 3.21: Visualization of the inter/intra-coding of the wavelet coefficients at the frames 2, 6, 38 and 39 of the Richter sequence. Red pixel: Y component inter-coded; Green pixel: U component inter-coded; Blue pixel: V component inter-coded; Black pixel: all component intra-coded; Other colors: several components inter-coded

3.6.1 Development Methodology

During this master thesis, ITICO has been developed with an iterative methodology. It is based on the software version of TICO written in C. After each iteration described below, the new leads to explore have been chosen in consultation with Pascal Pellegrin, hardware engineer at intoPIX and with the approval of the UCL members following this thesis.

- First iteration: during the first iteration, the investigation on which interprediction mechanism would suit the constraints of low latency and low complexity was carried out. It appeared that the simple matrix difference was the more fitting solution. The complexity remains low but the solution adds the need of a DDR buffer. The first concept of ITICO described earlier was implemented with the outcome that the use case in which this interprediction mechanism would be useful was desktop content thanks to its still content. As TICO has a quality problem at low bitrates on this kind of content, this interprediction mechanism seemed a good lead to tackle this issue.

- Second iteration: after analysing the results and the block diagram of the first concept of ITICO. The idea of moving the matrix difference came naturally as the Wavelet Transform is a linear operation. This change of place also opened the way to a first decision mechanism, a decision based on the sum of GCLI's of an entire precinct. This first prototype of decision allowed to mitigate the motion artifacts of moving objects but it also degraded the quality of still objects.

- Third iteration: in order to unlink moving objects from still objects, the decision mechanism was performed by subband. An improvement was observed but the concept needed to be pushed even further in a next iteration. This new decision led to a first refresh mechanism, forcing progressively each subband to be intra-coded regardless of the natural decision. This refresh mechanism allowed to deal with transmission error without the use of an I frame that would cause a flickering problem.

- Fourth iteration: the decision and the refresh mechanisms were improved to their final version described earlier. This final decision allowed to completely unlink the still objects from the moving objects whereas the final refresh mechanism is adapted to this new decision and takes advantage of the multiscale representation of the wavelet transform.

- Last iteration: a last problem remained unsolved by the previous iterations, the memory bandwidth required at the buffer in hardware implementations. Indeed, the hardware implementation of ITICO must remain a low cost solution but a DDR memory with a high memory bandwidth increases the price. Therefore, it was decided to compress the reference frame stored in the buffer. The normal TICO codec was the natural candidate to perform this compression. However, the complexity induced by adding a nearly complete TICO coder and decoder to ITICO was too important and therefore a simplified TICO was used for the referencee frame compression. Once the final codec complete, the trade-off in the different parameters of ITICO needed to be analyzed. After some tests, the default value of the parameters was decided. The refresh rate upper bound was set at 30 and the subband refresh rate was set to three times the numbers of groups of the subband. The upper bound of quality induced by a bitrate of 9bpp for the reference frame compression was judged visually good.

Chapter 4

Possible Improvement

Motion in videostreams is a big problem for the current ITICO that is not able to increase the quality of the compression for high motion sequences while the algorithm is twice as complex as the simple TICO.

4.1 Motion Prediction

A motion prediction mechanism could solve this problem but it would be at the expense of complexity and latency. It would require a good trade-off between performance and complexity and a reorganization of the data packing as well as modifications to some parts of the algorithm. Besides, as the difference is placed after the wavelet transform, the motion prediction mechanism should also be in the wavelet domain.

As the Wavelet transform is a linear operation, performing a Motion Prediction and Compensation in the wavelet domain as it would be performed in the spatial domain (each independent block has its own motion vector) is possible and the search window could be reduced to the different subbands but it would not exploit the multiscale representation of the Wavelet Transform.

However, there is a problem when exploiting this multiscale representation, the translation operator does not commute with the decimation operator (downsampling) thus coefficients of a translated image are not simply the translated coefficients. Indeed, if the signal is shifted by an even number, 2p, then the wavelet shift is p, that is alright. However, if the shift is an odd number, 2p+1, then the shift can not be predicted due to the decimation. Therefore, expanding the motion vector and reusing it at each resolution level provokes a drift that reduces the quality of the prediction. Happily, There exist several solutions to deal with this problem.

In [13], it was shown that for odd shifts the coefficients resulting of the lowpass filter can be approximated by interpolation quite efficiently thanks to the smoothness of the lowpass coefficient. The problem comes from the highpass coefficients that are not smooth and contains irregularities, an efficient interpolation is not possible. Therefore, they decided not to downsample the highpass coefficients with the justification that most of those coefficient are null and thus handled well by the entropy coding and quantization.

In [15], they decided to compute the two "phases" of the 1D wavelet transform, the first phase is when s^0 are the even indices and d^0 the odd indices (cfr Wavelet Transform description in chapter 1), the inverse is the second phase. They find the best combination of the phase and Motion Compensation Prediction and they keep this information as a small overhead.

In [16], they used another approach, they decided to compensate the different motions (translations, zooms,..) in consecutive frames by realigning them and then to perform a 3D wavelet transform on the 3D signal.

Many other approaches are described in several publications. Each technique mentioned here has its downsides and its advantages but it is worth noticing that none was conceived with the constraints of low latency and complexity. The first technique adds a lot more of coefficients to handle, the second requires to quadruple the number of wavelet transform (2D signal thus 4 possible phases) and the third is a complete change of paradigm compared to the current ITICO.

Conclusion

ITICO is a codec based on TICO that successfully uses interframe correlation in video sequences without an increase in latency and with a relatively low cost in complexity due to the simple nature of the interprediction mechanism: matrix difference between the input frame and the reference frame. In addition to this main element, several improvements allows to compensate the flaws of this interprediction mechanism.

The first flaw of the matrix difference is its inefficiency for video sequences with high motion. A process deciding which part of the frame should be coded with the interprediction scheme allows to handle this problem. A progressive refresh mechanism allows error resiliency without a high variation of quality between consecutive frames. Finally, compression of the reference frame before storage into a buffer reduces the required memory bandwidth of the buffer.

ITICO proves to be very efficient for desktop content compression at low bitrates (1-2bpp), a use case that was a weakness of TICO. The still nature of this video content allows the interprediction mechanism to reach a high quality of image.

However, ITICO does not provide any quality improvement to the videostreams with a lot of motion such as natural content or computer generated content. Happily, the TICO algorithm is already quite effective for this kind of content, even at low bitrates. Furthermore, a possible implementation of a motion prediction and compensation mechanism could help reaching even higher quality but at a high cost in complexity and latency.

Appendix A

Sequences frames

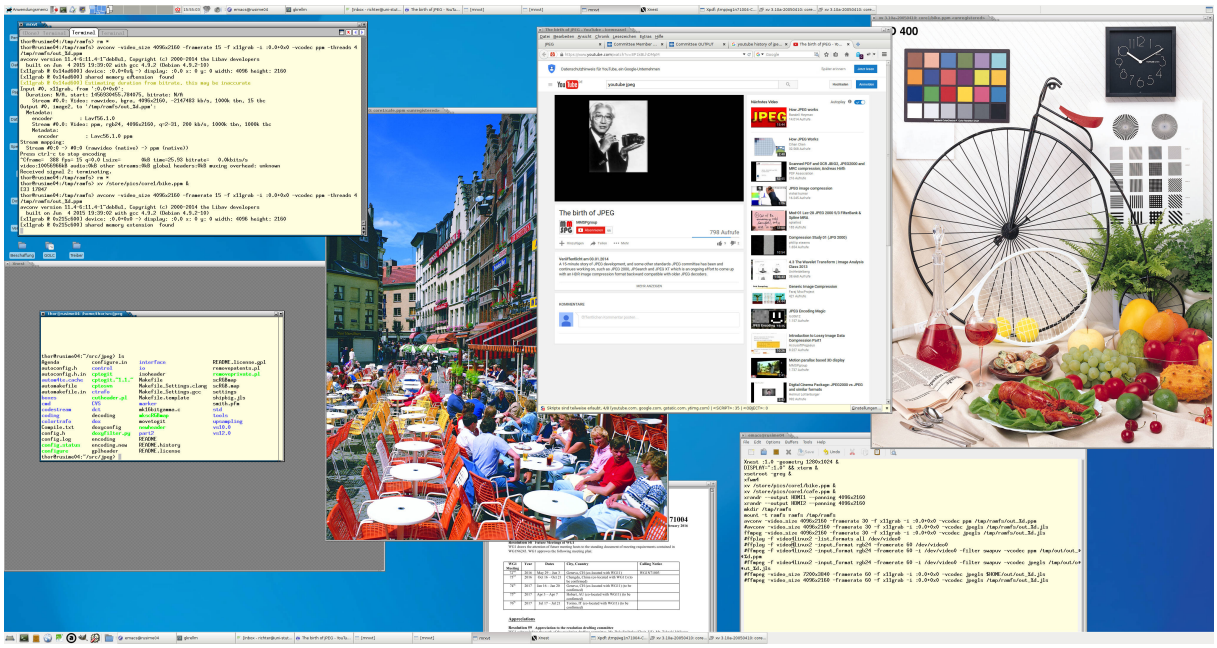


Figure A.1: First frame of the RICHTER ScreenContent 4096x2160p 15 8b sRGB 444 sequence

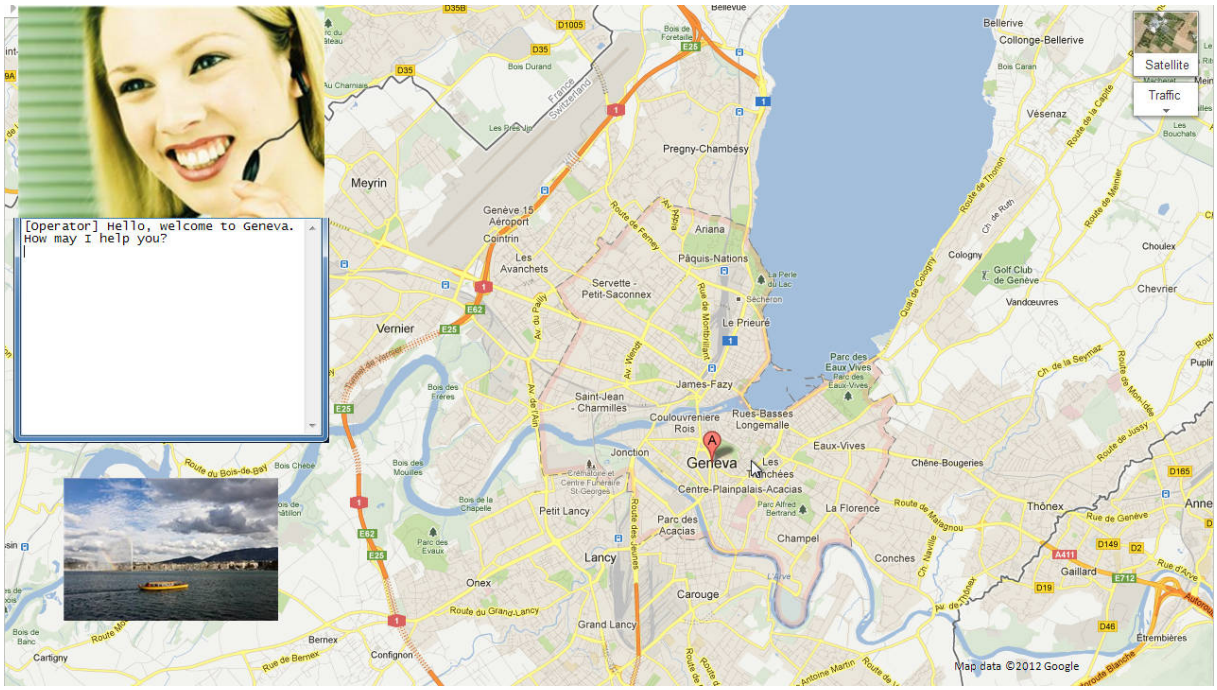


Figure A.2: First frame of the HUAWEI ScMap 1280x720p 60 8b sRGB 444 sequence

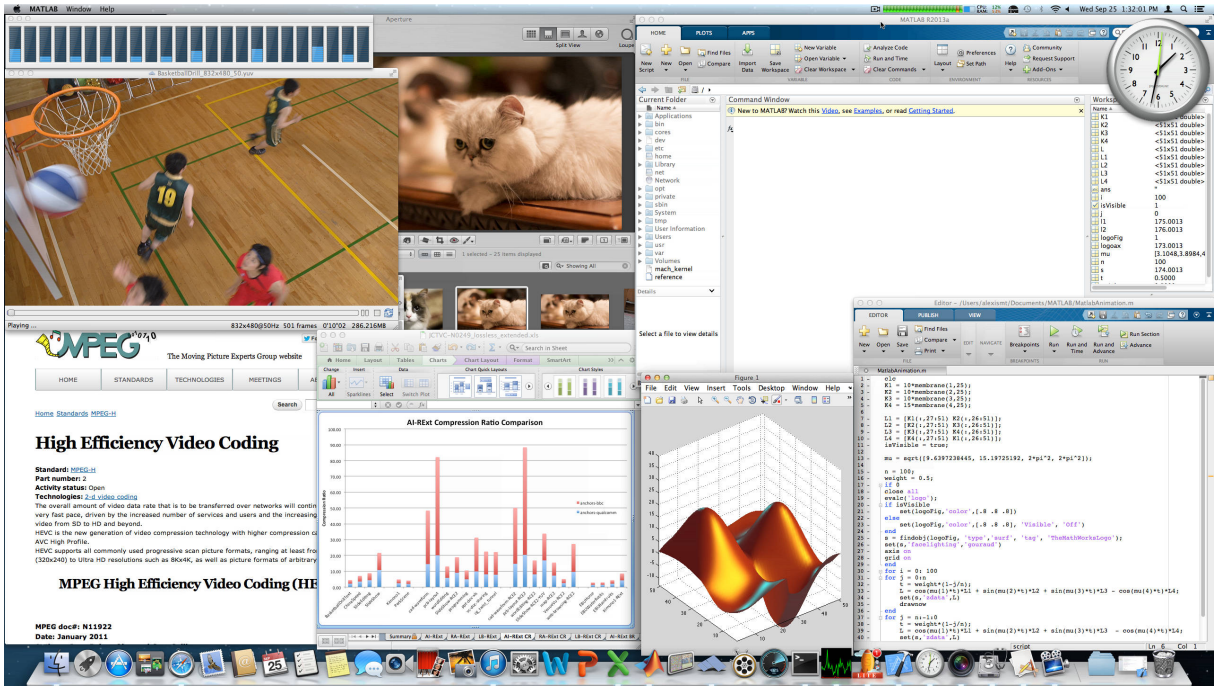


Figure A.3: First frame of the APPLE BasketBallScreen 2560x1440p 60 8b sRGB 444 sequence

Figure A.4: First frame of the VQEG CrowdRun 3840x2160p 50 8b bt709 444 sequence

Figure A.5: First frame of the VQEG ParkJoy 3840x2160p 50 8b bt709 444 sequence

Figure A.6: First frame of the EBU PendulusWide 3840x2160p 50 8b bt709 444 sequence

Figure A.7: First frame of the ARRI Lake2 2880x1600p 24 8b bt709 444 sequence

Figure A.8: First frame of the ARRI PublicUniversity 2880x1600p 24 8b bt709 444 sequence

Bibliography

[1] G. J. Sullivan and T. Wiegand, "Rate-distortion optimization for video compression," *IEEE Signal Processing Magazine*, vol. 15, pp. 74–90, Nov 1998.

[2] M. Rabbani and R. Joshi, "An overview of the jpeg 2000 still image compression standard," *Signal Processing: Image Communication*, vol. 17, no. 1, pp. 3 – 48, 2002. JPEG 2000.

[3] G. Qadir, X. Zhao, and A. T. Ho, "Estimating jpeg2000 compression for image forensics using benford's law," in *In SPIE Conference on Optics, Photonics, and Digital Technologies for Multimedia Applications*, 2010.

[4] I. Daubechies and W. Sweldens, "Factoring wavelet transforms into lifting steps," *J. Fourier Anal. Appl*, vol. 4, pp. 247–269, 1998.

[5] G. Impoco, "JPEG2000 - a short tutorial." `http://www.impoco.it/files/impoco_2004_tutorial_JPEG2000.pdf`, 2004.

[6] J. Louveaux, B. Macq, and O. Pereira, "INGI 2348 - information theory and coding." Course followed during academic year 2015-2016 at the Engineering School of Louvain.

[7] L. Jacques and C. D. Vleeschouwer, "LELEC2885 - image processing and computer vision." Course followed during academic year 2016-2017 at the Engineering School of Louvain.

[8] "JPEG XS website." `https://jpeg.org/jpegxs/index.html`. consulted on 08/05/2017.

[9] "HEVC website." `http://x265.org/`. consulted on 08/05/2017.

[10] A. Willème, "Overview of existing standards for low- latency and lightweight compression, benchmarking tools and results." JPEG XS workshop - Use cases for a low-latency lightweight image coding February 23rd 2016 – La Jolla USA.

[11] "Rdd 35:2016 - smpte - tico lightweight codec used in ip networked or in sdi infrastructures," *RDD 35:2016*, pp. 1–53, April 2016.

[12] "intoPIX codec submission for JPEG-XS call for proposals ." version 0.1.

[13] F. G. Meyer, A. Z. Averbuch, and R. R. Coifman, "Motion compensation of wavelet coefficients with wavelet packet based motion residual coding," 1997.

[14] A. Nosratinia and M. T. Orchard, "Multi-resolution backward video coding," in *Proceedings of the 1995 International Conference on Image Processing (Vol.2)-Volume 2 - Volume 2*, ICIP '95, (Washington, DC, USA), pp. 2563–, IEEE Computer Society, 1995.

[15] X. Li, L. Kerofsky, and S. Lei, "All-phase motion compensated prediction in the wavelet domain for high performance video coding," in *Proceedings 2001 International Conference on Image Processing (Cat. No.01CH37205)*, vol. 3, pp. 538–541 vol.3, 2001.

[16] D. Taubman and A. Zakhor, "Multirate 3-d subband coding of video," *IEEE Transactions on Image Processing*, vol. 3, pp. 572–588, Sep 1994.

[17] A. Descampe, F. O. Devaux, G. Rouvroy, J. D. Legat, J. J. Quisquater, and B. Macq, "A flexible hardware jpeg 2000 decoder for digital cinema," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 16, pp. 1397–1410, Nov 2006.

Rue Archimède, 1 bte L6.11.01, 1348 Louvain-la-Neuve **www.uclouvain.be/epl**