Universidade de Aveiro Departamento de Electrónica, Telecomunicações e Informática,
2019

Miguel Oliveira Inocêncio

Co-processador da Transformada para AV1

AV1 Transform Co-Processor

DOCUMENTO PROVISÓRIO

Miguel Oliveira Inocêncio

Co-processador da Transformada para AV1

AV1 Transform Co-Processor

Dissertação de Mestrado apresentada à Universidade de Aveiro, para obtenção do grau de Mestre em Engenharia Eletrónica e de Telecomunicações, sob orientação do Professor Doutor António Navarro ...

DOCUMENTO PROVISÓRIO

o júuri / the jury

presidente / president

ABC
Professor Catedrático da Universidade de Aveiro (por delegação da Reitora da Universidade de Aveiro)

vogais / examiners committee

DEF
Professor Catedrático da Universidade de Aveiro (orientador)

GHI
Professor associado da Universidade J (co-orientador)

KLM
Professor Catedrático da Universidade N

agradecimentos / acknowledgements

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum. . .

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris. . .

Palavras-Chave HEVC, ...

Resumo

Nulla malesuada porttitor diam. Donec felis erat, congue non, volutpat at, tincidunt tristique, libero. Vivamus viverra fermentum felis. Donec nonummy pellentesque ante. Phasellus adipiscing semper elit. Proin fermentum massa ac quam. Sed diam turpis, molestie vitae, placerat a, molestie nec, leo. Maecenas lacinia. Nam ipsum ligula, eleifend at, accumsan nec, suscipit a, ipsum. Morbi blandit ligula feugiat magna. Nunc eleifend consequat lorem. Sed lacinia nulla vitae enim. Pellentesque tincidunt purus vel magna. Integer non enim. Praesent euismod nunc eu purus. Donec bibendum quam in tellus. Nullam cursus pulvinar lectus. Donec et mi. Nam vulputate metus eu enim. Vestibulum pellentesque felis eu massa.

Quisque ullamcorper placerat ipsum. Cras nibh. Morbi vel justo vitae lacus tincidunt ultrices. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. In hac habitasse platea dictumst. Integer tempus convallis augue. Etiam facilisis. Nunc elementum fermentum wisi. Aenean placerat. Ut imperdiet, enim sed gravida sollicitudin, felis odio placerat quam, ac pulvinar elit purus eget enim. Nunc vitae tortor. Proin tempus nibh sit amet nisl. Vivamus quis tortor vitae risus porta vehicula.

Keywords HEVC, ...

Abstract

Nulla malesuada porttitor diam. Donec felis erat, congue non, volutpat at, tincidunt tristique, libero. Vivamus viverra fermentum felis. Donec nonummy pellentesque ante. Phasellus adipiscing semper elit. Proin fermentum massa ac quam. Sed diam turpis, molestie vitae, placerat a, molestie nec, leo. Maecenas lacinia. Nam ipsum ligula, eleifend at, accumsan nec, suscipit a, ipsum. Morbi blandit ligula feugiat magna. Nunc eleifend consequat lorem. Sed lacinia nulla vitae enim. Pellentesque tincidunt purus vel magna. Integer non enim. Praesent euismod nunc eu purus. Donec bibendum quam in tellus. Nullam cursus pulvinar lectus. Donec et mi. Nam vulputate metus eu enim. Vestibulum pellentesque felis eu massa.

Quisque ullamcorper placerat ipsum. Cras nibh. Morbi vel justo vitae lacus tincidunt ultrices. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. In hac habitasse platea dictumst. Integer tempus convallis augue. Etiam facilisis. Nunc elementum fermentum wisi. Aenean placerat. Ut imperdiet, enim sed gravida sollicitudin, felis odio placerat quam, ac pulvinar elit purus eget enim. Nunc vitae tortor. Proin tempus nibh sit amet nisl. Vivamus quis tortor vitae risus porta vehicula.

Contents

List of Figures

List of Tables

Acronyms

AOM Alliance for Open Media.

AV1 AOM Video 1.

CABAC Context Adaptative Binary Arithmetic Coding.

CMOS Complementary metal–oxide–semiconductor.

Codec Encoder-Decoder.

CRT Cathode Ray Television.

DCT Discrete Cosine Transform.

DFT Discrete Fourier Transform.

FFT Fast Fourier Transform.

fps frames per second.

GPU Graphical Processing Unit.

HEVC High Efficiency Video Coding.

IC Integrated Circuit.

JVT Joint Video Team.

MPEG Motion Picture Experts Group.

TV Television.

UHD Ultra-High-Definition.

VLC variable length codes.

WGN White Gaussian Noise.

WHT Walsh-Hadamard Transform.

Glossary

Codec Encoder-Decoder. Also referred to the method of compressing and decompressing a video sequence.

H.264/AVC Previous state of the art video codec from Joint Video Team (JVT), released in 2007. As to the writing of this work, it is the most used video compression algorithm..

Interlaced scanning Technique used by televisions for broadcasting and displaying, where only odd or even numbered lines of a frame are transmitted/displayed at a time, alternately.

JPEG Still image compression format, developed by the Joint Photographic Experts Group (JPEG).

libaom Reference software for AV1, released by Google in June 2018.

Pixel Picture Element.

Progressive scanning Technique used by more recent screens, where each frame is displayed as a whole, from top to bottom, and left to right.

RGB Color space based on the addition of Red, Green and Blue components for complex color representation.

VP8/VP9 Open-format video codecs developed by Google, released in 2008 and 2013, respectively.

Todo list

CHAPTER 1

Introduction

1.1 | Background and Motivation

Since the spark of television research in 1887, a tremendous investment has been put into increasing the quality of images, cameras and screens that display them [1].

In the early years of mechanical television, this desire was pursued by making changes to the *Nipkow* disks, up to the decline of the mechanical TV, around the 1930's. The consequential rise of all-electronic TVs started with the capture of images with the same cathode tubes put into Cathode Ray Televisions (CRTs), with broadcasts of the live analog recordings, since there were no available methods of storing images, up to 1955, with the development of the open-reel magnetic tape [2].

The evolution of Complementary metal–oxide–semiconductor (CMOS) technologies however, led to the downfall of cathode ray tubes, and to the rise of image capture to a digital sensor, that allowed better image captures and lower demands in terms of physical storage space. However, with the desire for higher fidelity video, the quantity of information captured also increased. Whether by increasing the sensor resolution, color bit depth or frame rate, the captured video sequences have increased its size throughout the years. For instance, for a video of 640×360 (considered as a low resolution), at 30 frames per second (fps), considering each captured color (RGB) is represented with 8 bits, there is approximately 166 Million bits per second (Mbps) of captured information. This means that a short 5 minute video would occupy more than 6 Giga Bytes (GB) of memory. This aspect gets more severe once higher resolutions are considered. For newer standards such as 4K Ultra-High-Definition (UHD) (3840×2160) or 8K UHD (7680×4320), under the same conditions, a ten minute video would occupy 448 GB and 1792 GB of raw data, respectively.

To further aggravate the situation, video consumption got massively adopted on the average consumer level, and continues to grow, both in the average number of hours watched by users and in the resolutions of the video, making the bandwidth used on the visualization of video footage the highest between all other application. With the development of higher video sizes, increase of the average number of connected devices per user and overall market expansion through the number of consumers, this margin will continue to grow [3, Trends 1 & 4].

This problem has led to the introduction of a new concept: *Video Compression* [1], which is the process of reducing the size of a video sequence, while still maintaining its playback capabilities. The Codec takes advantage of redundant information present on the raw data to reduce the size of the video, without heavily modifying the original picture or its quality.

The first form of video compression, Interlaced scanning, dates from 1940, and was purely

[1] Also called *Video Coding*.

analog. This solution was introduced with the intent of reducing the necessary broadcasting bandwidth for old CRTs, without decreasing the displayed fps. And even though this technique has been implemented over more than seventy years, it has proven to be so efficient that most TV channels today still use interlaced broadcasting.

However, analog television is now obsolete, as well as CRTs. The massive developments in Integrated Circuit (IC) fabrication led to the rise of the digital era we now live in. Therefore, most screens (be it televisions, monitors or cellphones) use digital, Progressive scanning. As such, the use of analog compression techniques wasn't applicable. Accordingly, the evolution of digital video led to the development of digital compression techniques, such as the one presented in this work.

Being purely digital, these methodologies rely on computers and other processors to analyze data and apply the compression algorithms, making them very demanding processes from a computational standpoint. As expected, a high compression ratio is only obtainable by a high complexity algorithm, which also increases with the size of the video (more data leads to more analysis). Since in the early days of digital video, the used resolutions were lower as to the ones used in the present days, the compression algorithms used were not very demanding. However as the pursuit for higher quality video continued, so did the necessity for better compression ratios, and therefore the computational needs also increased. Such complex softwares lead to a high power consumption from the processor executing it, making such implementations unsuitable for portable, battery limited applications, such as cellphones or laptops. Besides this huge factor, such softwares tend to be very slow, specially when a real time compression or decompression is desired.

To amend for these factors, and to increase the reachability of high quality video to as many users as possible, these applications needed to have a viable solution that didn't compromise its usability. Accordingly, a new approach has been implemented on the most recent codec's. Besides the optimization of pure software compression/decompression solutions, there has been a great focus on the development of specialized hardware for such codecs. This solution could redress many of the problems presented previously, making them viable on a mobile implementation, as well as other specialized appliances, since such co-processors usually present a better performance than generic ones. This tendency has already been verified on the implementation choices on recent smartphones [4, p. 14], as well as recent *Nvidia* Graphical Processing Unit (GPU) lineups [5].

Since each compression algorithm tend to be very different from its predecessors, either by making changes to its bitstream or functioning principles, each time a new codec is released, there is a need to backup its development with a new set of hardware implementations. This makes the improvement of video compression techniques a continuous effort, in many engineering branches, as the technology needs to keep up with the demands of consumers, in a variety of applications.

Due to the broad access to video, and its influence in a variety of markets (besides video consumption itself), big companies have made investments on the improvement of video quality, and respective compression algorithms. These investments have provoked somewhat of a *"Codec War"*. Since 2010, several video codecs have been deployed, and quickly replaced by a newer version, which presents better compression gains, at a lower quality degradation, such as the replace of *VP8* (released in 2008) with *VP9* (2013).

1.2 | Scope

AOM Video 1 (AV1) is the most recently released[2] video codec. It was developed as a Joint Development Foundation [6] project, under the name of Alliance for Open Media (AOM) [3]. This codec took the same objective as its main predecessor, *VP9*, which was to be an open source, royalty free alternative to Motion Picture Experts Group (MPEG)'s state of the art video codec, *High Efficiency Video Coding (HEVC)*.

Upon release, *VP9* rivaled *HEVC's* performance. However, soon after, the market demanded higher compression performance, giving origin to consortium of enterprises that now represent AOM, and to the development of *AV1*, in 2015. The first release of this coding format was made in March 2018, with the first release of its reference software, *libaom*, being made three months later, in June 2018.

Besides its main objectives, *AV1* was also developed with the intent of being implementable in hardware. Therefore, various design choices were made to make the algorithm low memory consuming, and highly parallelizable.

The desired compression performance was obtained at the cost of a highly complex algorithm (and reference software), that severely outperforms *VP9*, at the cost of much higher compression times [7].

Taken these factors, there is a high demand for dedicated hardware architectures, that can speed up the compression/decompression times and reach real-time usability on live-streaming applications, such as video-conferencing, live-content visualization, etc.

*Thesis objectives

1.3 | Outline

*General outline of the different chapters **Last section to do**

[2] Currently, there are other codecs being developed, without official bitstream release

[3] Further explained in Chapter 2

References

[1] Mark Schubin. "What Sparked Video Research in 1877? The Overlooked Role of the Siemens Artificial Eye [Scanning Our Past]". In: *Proceedings of the IEEE* 105.3 (Mar. 2017), pp. 568–576. ISSN: 0018-9219, 1558-2256.

[2] Marco Jacobs and Jonah Probell. "A Brief History of Video Coding". en. In: (), p. 6.

[3] *Cisco Visual Networking Index: Forecast and Trends, 2017–2022 White Paper*. en. https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/white-paper-c11-741490.html.

[4] Scientiamobile. *Mobile Overview Report April – June 2018*.

[5] *Video Encode and Decode GPU Support Matrix*. en. https://developer.nvidia.com/video-encode-decode-gpu-support-matrix. Nov. 2016.

[6] *Joint Development Foundation*. en. http://www.jointdevelopment.org/.

[7] Dan Grois, Tung Nguyen, and Detlev Marpe. "Performance Comparison of AV1, JEM, VP9, and HEVC Encoders". In: *Applications of Digital Image Processing XL*. Ed. by Andrew G. Tescher. San Diego, United States: SPIE, Feb. 2018, p. 120. ISBN: 978-1-5106-1249-5 978-1-5106-1250-1.

CHAPTER 2

Video Compression Systems

2.1 | Basic Principles

Video Compression Systems have been in development for approximately forty years, with the first video codec, *H.120*, being released in 1984. It was composed of basic operations, which didn't correlate to good compression performances. This has lead to a quick downfall of its usage, being aggravated by the release of the *H.261* standard by 1984.

However, the building blocks on which later standards were based are the same as in the first generations, i.e., the strategies implemented on newer standards exploit the same *redundancies* as previous, less efficient, codecs.

By redundancies, it is meant disposable information to the playback of an image sequence. This concept is the key of video compression. Throughout the years, the enhancement of video codecs was based on the improving the algorithms which can reliably represent a video, while maintaining the least of the original information. In other words, the video sequence is analyzed for predictable/identifiable characteristics, e.g. the movement of a subject or the edge of an object, calculates strategies of predicting nearby pixel values through that information and disposes the unused information. This process is mentioned as *redundancy removal*.

This way, to have a better understanding of the functioning behind video codecs, the mentioned redundancies, and respective origins, are presented. Most of such have origin on the way humans perceive vision, being this the first topic of this chapter.

2.1.1 | Human Visual System

Most of the compressed/decompressed video nowadays is directed to content visualization by consumers, with the exception of some network-driven image processing applications, such as automatic video surveillance. Therefore, the compression of video sequences has the intent of making changes to the original data, without serious impact to the users' perception. This process is mentioned as the removal of the *Psychovisual redundancy* [2]. Therefore, a basic understanding of the visual system can clarify many of the design choices made in video compression applications, and why their use doesn't present much impact on the quality of the image, while greatly reducing its size.

The image perception starts in the human eye, represented in figure 2.1. Its different constituents accomplish different tasks, from focusing, to aperture control. Although their importance to the overall functioning of the eye, the part that matters most to the focus of this work is the innermost membrane, the retina.

Once the desired image is properly focused by the lens, an inverse version of it is shined on

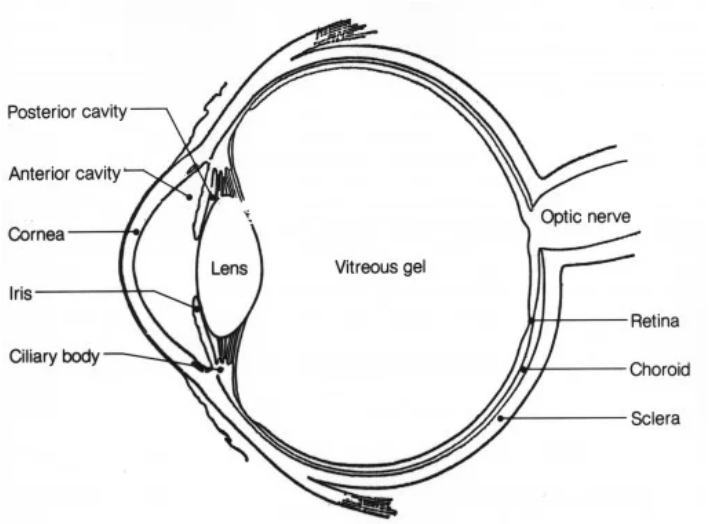


Figure 2.1: Representation of an human eye [3]

the aforementioned membrane, which is covered by two types of light sensitive cells, the *cones* and *rods*, which transform the observable image into a series of pulses, that get subsequently processed.

The cones are highly sensitive to color, being responsible for the *photonic* or *bright-light* vision. There are three different types of cones, corresponding to the wavelength they are susceptible to. These are the S, M and L cones, being sensitive to, approximately, the blue, green and red light, respectively, making a somewhat similar capture to the RGB color system.

On the other side, rods aren't stimulated by bright light, being more active on low illumination levels. This aspect makes them responsible for giving a rough overview of the field of view. This is called as *scotopic* or *dim-light* vision. These cells are spread more broadly across the retina, while to the cones, which is also observable in the number of cells (approximately 6 million cones, to 100 million rods).

From this, it's already observable that the human visual system is more sensitive to differences on the luminosity, than to the color of an object [4], which is a starting point for compressing video, as will be shown later in this chapter. However, many other opportunities come from the processing of the nerve signals, and the *psychovisual* perception that follows.

Although more sensitive to *luminance*, there is a threshold to which the difference between two objects — ΔI — can't be discerned. This relation is mentioned as *contrast sensitivity function*, which is roughly approximated with the *Weber's Law*

$$\frac{\Delta I}{I} \approx constant \tag{2.1}$$

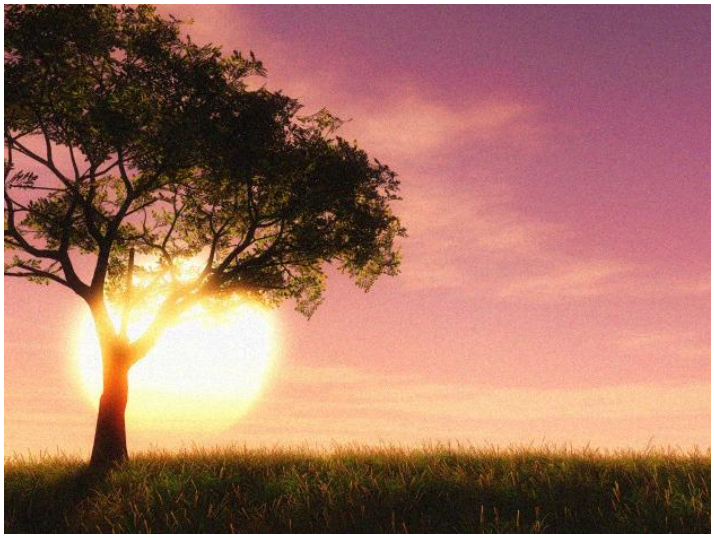
Analyzing this equation, it's possible to conclude that the darker an object is, the lower the difference in luminance needs to be to distinguish another object. Also, darker images tend to be more susceptible to compression artifacts.

Besides the luminance values, the spatial and temporal frequencies also represent an important role in the perception of such errors.

The image 2.2 gives an example of the dependency with spatial frequency. The first image 2.2a represents the original image, which got added with White Gaussian Noise (WGN),

represented in figure 2.2b. As it is observable, these artifacts are less noticeable on the highly detailed areas (branches and leafs of the tree) than in the smooth ones (sky in the top right corner). The effect of *Weber's law* is also observed if we analyze the effect that the white noise as in the bright sun area, when compared to the darker areas.

(a) Original Image [5]



(b) Image with added WGN

Figure 2.2: Example of the effect of added noise on figure

Temporal frequency dependency, although more challenging to exemplify, is easily understandable. On a sequence of frames with fast camera, or subject movements, the human eye doesn't have the ability to track details or other artifacts, while in slow moving scenes, it can easily identify errors.

These are some of the "*flaws*" of the human visual system, that get exploited during the compression of video. However, other *redundancies*, inherent from the captured images themselves contribute to the reduction of the video size, as will be described in the following

sections.

2.1.2 Redundancy Exploitation

Even though there are countless observable subjects and sceneries, it's unfair to think of a frame as a random sequence of pixels. Objects tend to represent clusters of pixels with roughly the same values, moving objects follow predictable directions, etc. Such characteristics represent redundancies that can be removed.

2.1.2.1 Spatial Redundancy

Spatial redundancy comes from the similarity between neighboring pixels, on one frame. This aspect is easily verified through the autocorrelation of an image, as will be shown in the following example.

Taking image 2.2a and calculating its autocorrelation with various horizontal shifts, gives origin to the graph in image 2.3.

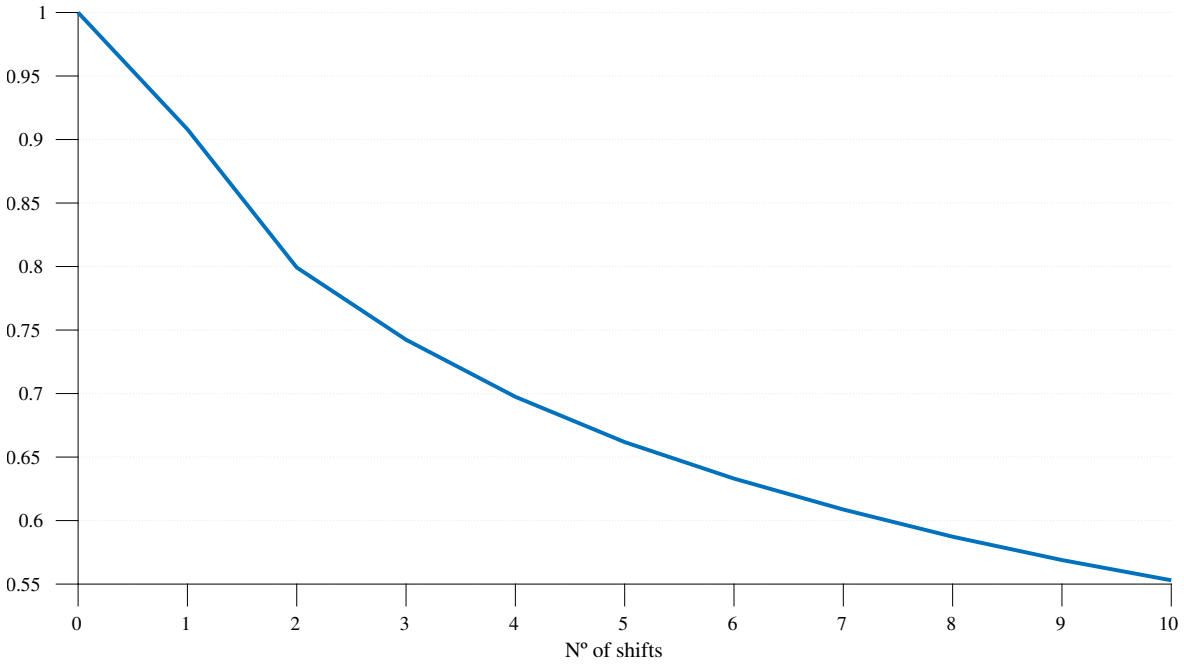


Figure 2.3: Autocorrelation of image 2.2a, with horizontal shifts

As it is observable, for shorter shifts, the normalized autocorrelation is very close to 1, since most of the de-correlation comes for the mismatching edges. Although this relation varies depending on the image, it's safe to assume that it is very similar for the majority of the cases.

Such study gives a promising opportunity for compression, since it means that most pixels can be predicted from its neighbors. This aspect as lead to what now is known as *differential* or *predictive* coding.

On a video compression system, the spatial redundancy is considered in the *intra-prediction* block, which calculates pixels, or pixel blocks, through its surrounds.

2.1.2.2 | Temporal Redundancy

As expected, a series of consecutive frames, on the same subject, tend to be very similar between each other, especially if considered the 30 or 60fps desired nowadays.

Making a similar analysis to what was made in section 2.1.2.1, a series of frames of the *Stefan* sequence [6] was considered, and the cross correlation between the first and the following nine was calculated, giving origin to the graph in figure 2.4.

verify graph generation

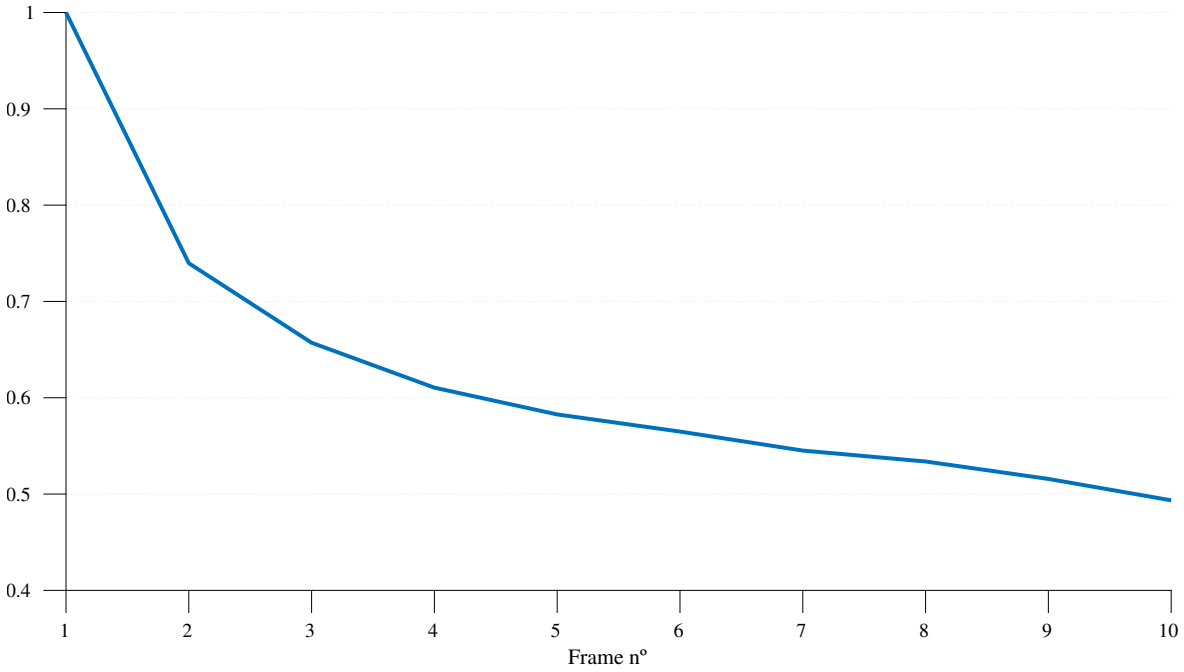


Figure 2.4: Cross-correlation between the first and following nice frames of the *Stefan* sequence

Similarly to what happened in the previous example, the cross correlation between consecutive frames is very high. Even though for faster moving scenes this relation might not be as pronounced, its application on video coding greatly contributes to the compression verified in the latest codecs.

The codec takes advantage of this redundancy in the *inter-prediction* stage, which is composed by the *Motion Estimation* (MC) and *Motion Compensation* (ME) blocks. On this stage, blocks of pixels in nearby frames are analyzed for movement, predicting its position for following frames.

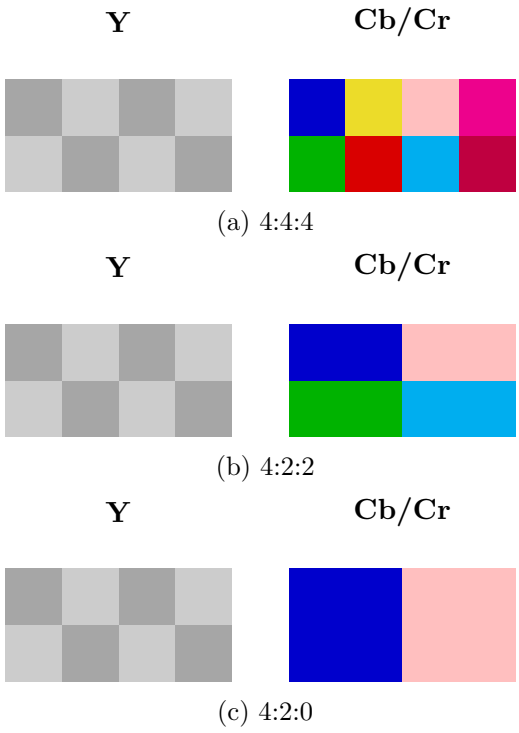
2.1.2.3 | Psychovisual Redundancy

From the redundancies presented in section 2.1.1, video codecs implement compression measures in various stages.

The first measure is the *chroma subsampling*, which takes advantage of the lower perception to color, discarding some of the *chroma* samples, depending on the subsampling chosen.

Typically, a pixel value is represented in one luminance and two chrominance values, on the *YCbCr* color space. The subsampling is defined in through the relation of luminance to

chroma samples, being the most common the 4:4:4, 4:2:2 and 4:2:0 standards, represented in figure 2.5. In the first one, no chroma samples are discarded, which means that for each 4 luminance (Y) samples, there are 4 Cb and 4 Cr samples. Correspondingly, in the second standard, for each 4 Y samples, only 2 of each color components are maintained. The last example, although its misleading term, means that only 1 of each 4 chroma samples are kept.



(a) 4:4:4

(b) 4:2:2

(c) 4:2:0

Figure 2.5: Representation of chroma subsampling

From the reduced sensitivity to details (or areas with high spatial frequency), the compression is explored in the *Transform* (T) and *Quantization* (Q) blocks. In the first stage, blocks of pixels are evaluated in their frequency components. These are then evaluated in the second stage, where the least significant ones get discarded. In the decoder, the image is reconstructed with the maintained coefficients, without much impact to the image quality. This process is further explained throughout the work.

On the Quantization block, some work was also developed to account for *Weber's law*, where the quantization depends on the average luminance value of the block. This concept was first introduced in [7], and since then experimented in various codecs, such as HEVC [8].

2.1.2.4 | Coding Redundancy

Coding redundancy is directed to the method of representing information in the digital domain, i.e., the bits themselves, and how they are organized.

It is known that symbol probability plays a major role in information compression, across a wide variety of branches, and video is no exception. Taking this into account, codecs take advantage of coding redundancy in the *Entropy Encoder* stage.

2.1.3 | Basic Video Compression/Decompression System

From the basic principles of the previously mentioned blocks, it is possible to integrate them into two complete compression — *Encoder* — and decompression — *Decoder* — modules.

2.1.3.1 | Encoder Model

The encoder's objective is to compress a video sequence, turning it into a readable *encoded bitstream*. To do this, the previously presented strategies get implemented on a system based on the schematic of figure 2.6.

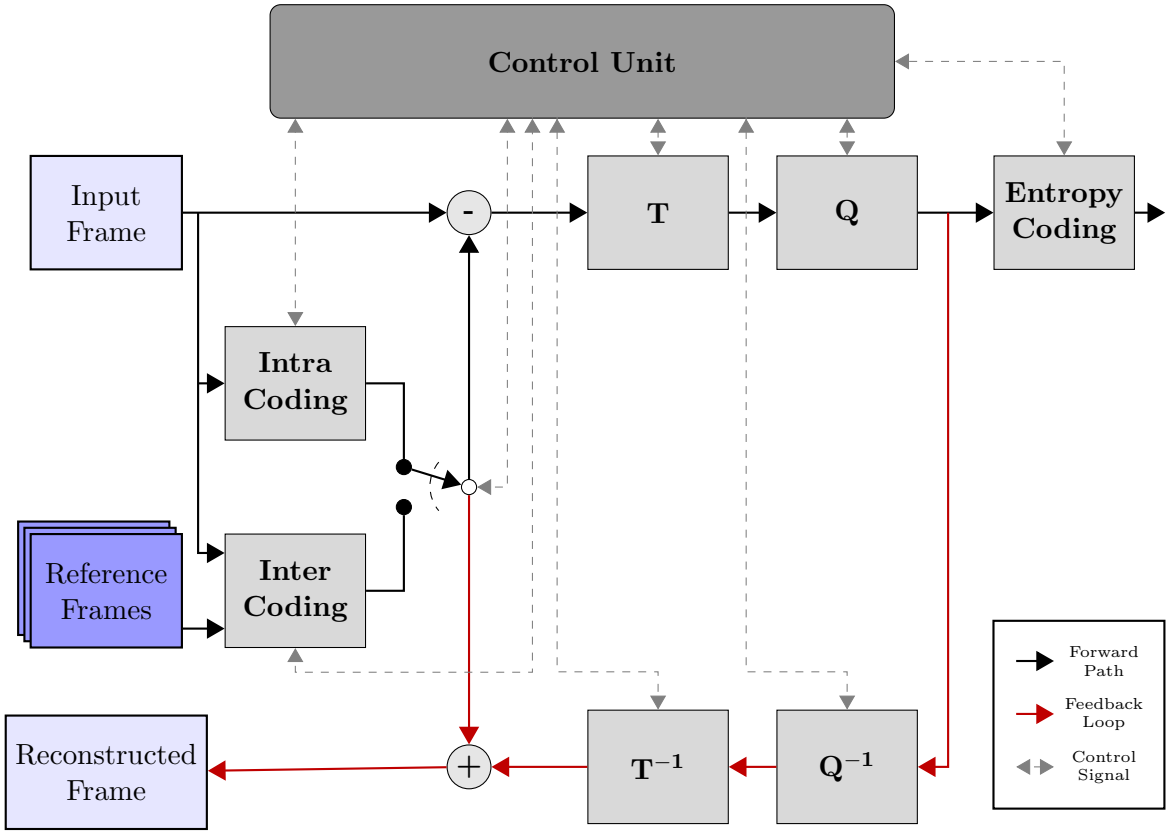


Figure 2.6: Simplified Basic Encoder Model

The encoding process starts on the *Input Frame*, which can be of two types. *I Frames* are encoded using only the information present in themselves, i.e., using only *Intra Prediction/Coding*, while *P Frames* may use predictive coding from previously encoded frames [1].

The input gets split into blocks, which get fed into the two main blocks of a video encoder: the *Intra* and *Inter* Prediction blocks.

The *Intra Coding* block, as mentioned previously, deals with the spatial redundancy, by predicting the current block from the pixels above and to the left of its upper and left edges. The prediction may be done with various algorithms, ranging from calculating the average

[1] Most video codecs allow that the encoding sequence isn't the same as the temporal sequence. This allows to reference frames displayed next to the one being encoded.

from the reference pixels, to replicating these according to a certain direction. One such example is presented in figure 2.7, where pixels B through H get spread across a 4×4 block, diagonally.

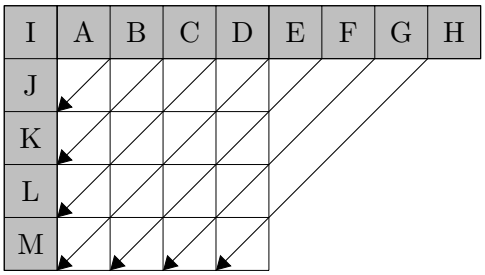


Figure 2.7: Directional Intra-prediction example

Into the *Inter Coding* block, go two inputs. The currently encoding block, as well as a bank of previously encoded frames, named *Reference Frames*. Firstly, the frames inside the buffer get searched for blocks resembling the former input. Once found, this process generates a *motion vector*, corresponding to the difference between the position of the block found in the reference frame, and the position of the currently encoding block, as shown in figure 2.8.

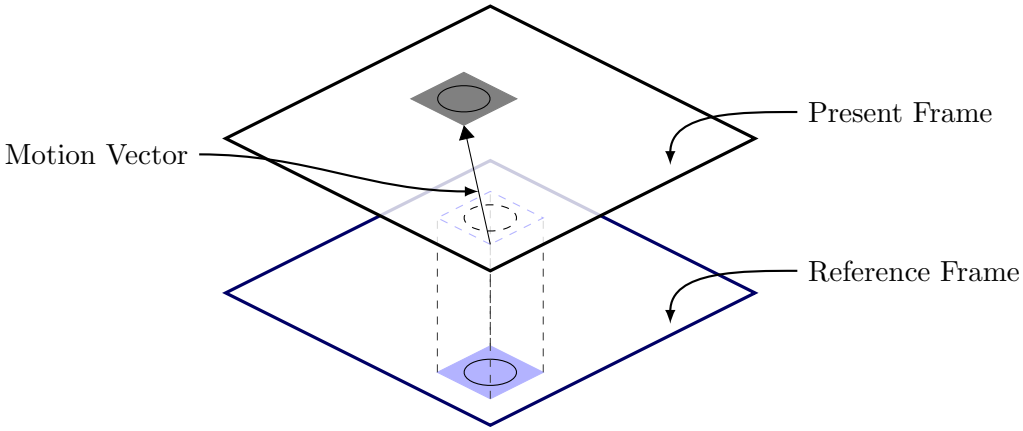


Figure 2.8: Inter-prediction example

After the prediction stage, the chosen output between the two processes, i.e., the predicted block, gets subtracted by the current one, giving origin to the *residue*. This corresponds to the pixel value differences between the original and predicted blocks. Lower *residues* indicate more efficient prediction stages.

The next stage, the *Forward Transform*, is the focus of this work. It takes the residue blocks, which may not be the same size of the prediction blocks, and evaluates them according to its spatial frequencies. Its output corresponds to a series of *coefficients*, that are related to the similarity — or *correlation* — between the input block and a series of *basis images*. This process is further explained in chapter 3.

On the *Quantization* stage, the coefficients calculated in $\mathbf{T}$ get scaled according to a *Quantization Matrix*. This stage takes advantage of the eye's lower perception to high frequency details, and scales the higher frequency coefficients by a higher value, than the lower, more significant ones. In most of the transformed blocks, this leads that only a few low frequency

components are maintained, while the others get nullified, since they are not relevant to the reconstruction of the image. Therefore, this stage is the the one that presents the higher loss, although the previously presented also introduce errors.

The wipe out of the least significant coefficients is particularly efficient when paired with the last stage before the output, the *Entropy Encoder*. On this block, **Q**'s output block gets run sequentially via a *zig-zag scan*, which first passes through the lower frequency coefficients, followed by the higher frequency ones.

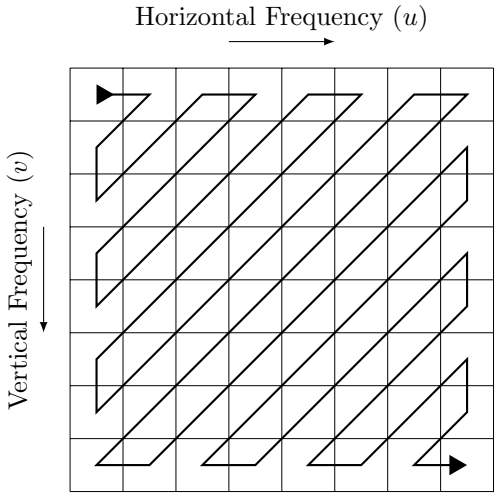


Figure 2.9: Demonstration of Zig-Zag Scan

In most of the cases, this causes that the non-zero coefficients get read first, followed by a sequence of zeros. Such sequence benefits heavily of being encoded with variable length codes (VLC), such as *Huffman Tree Codes* or Context Adaptative Binary Arithmetic Coding (CABAC). Off all the processes, this is the one that doesn't introduce further distortion into the encoded sequence, which is the reason it doesn't get included in the *feedback loop*.

The intent of this loop is to get an exact same copy of the frame reconstructed in the decoder. This reconstructed frame gets used as the reference for intra-prediction, or gets put into the reference frame buffer to be used in a later inter-prediction process.

The output of the encoder is the quantized coefficients, as well as the necessary information to recreate the encoded blocks, such as the type of prediction used, the transformation *kernel* [see p.18], quantization matrix, et al. These encoding parameters are the choices made by the *Control Unit*, which although represented by a block in figure 2.6, may not be a local process, independent from all others.

Since *H.264*, most video codecs standardize the decoding process, specifically, the allowed tools for compressing the video, and how to use them. This means that the encoding process is widely adaptable to the compression objectives, as long as the final product is a bitstream following the norms set on the codec's standard [9]. Therefore, the definition of a *Control Unit* is ambiguous in this context, since such unit can simply represent a set of parameters to be used throughout the encoding process[2], or an algorithm that can change between the different capabilities of the codec, in order to achieve an objective, such as a specific distortion

[2]One such example would be *lossless* compression modes, which use very a concise conditions on each stage, in order to get the least distortion.

rate, or not surpass a maximum bit rate.

2.1.3.2 | Decoder Model

As expected, the decoder (figure 2.10) does the backwards operation of the encoder on figure 2.6. It starts by analyzing the bitstream, separating the control information from the encoded and quantized coefficients.

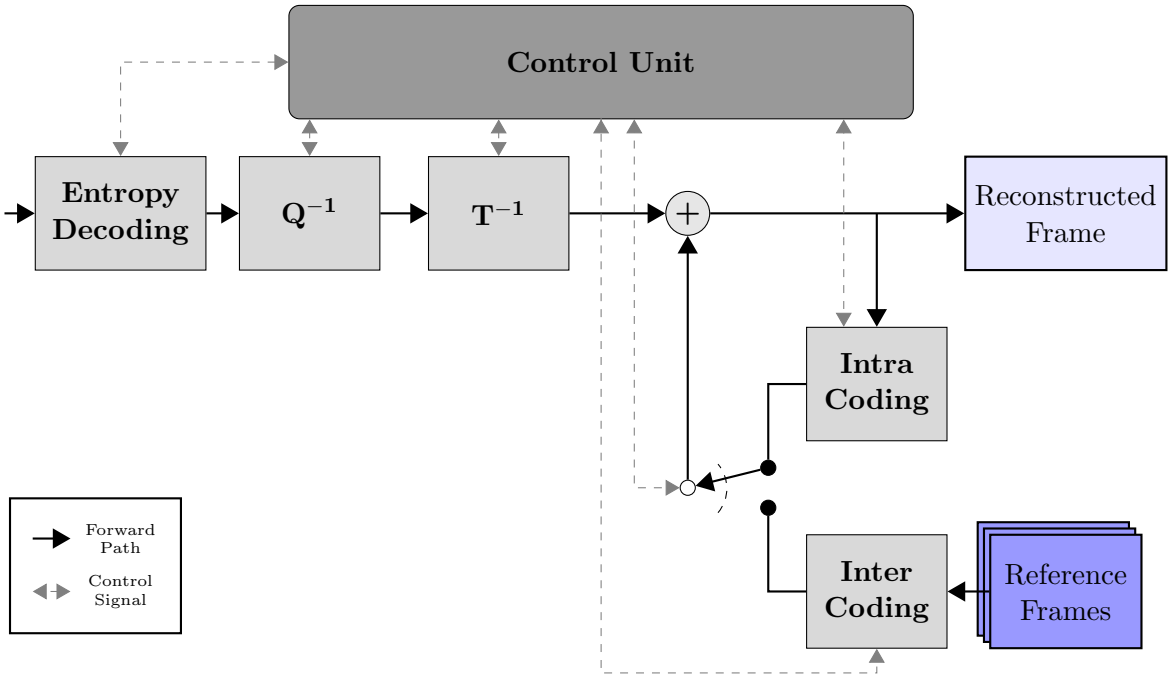


Figure 2.10: Simplified Basic Decoder Model

Having the encoding choices performed by the encoder, the decoder returns the *coding redundancy* to the quantized coefficients, on the *Entropy Decoding* stage. This corresponds to a translation from the varying length code used in codification, back into the raw coefficients.

The *Inverse Quantization* rescales the maintained coefficients, resulting from the previous *Quantization* stage. With this, it is meant that the same quantization matrix used when dividing the transformed coefficients, in the encoder, is now multiplied by the quantized parameters. It must be kept that this operation does not output an exact copy of the transformation coefficients, as a lot of information is permanently lost in **Q**. This process can be seen in figure 2.11.

As can also be seen in this figure, the *Inverse Transform* converts the coefficients back into spatial coordinates, therefore getting the restored residue. To obtain the final approximation of the block being decoded, this residue must be added to the same predicted block from the encoder. To do so, the *Intra* or *Inter Prediction* stages act according to the choices made in the encoding process, as to regenerate this block.

In the decoder, the *Control Unit* represents the process that organizes the different stages, according to the choices done in the encoding stage.

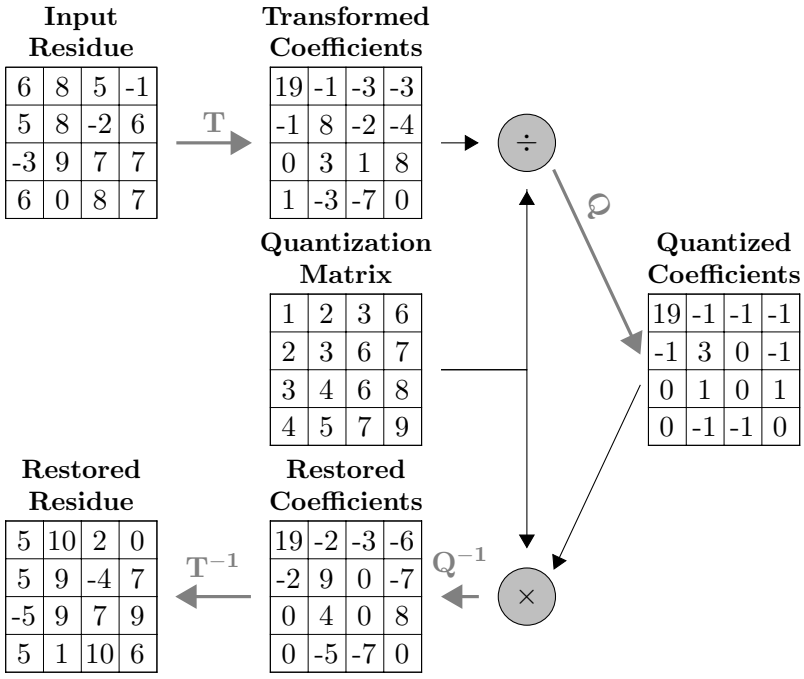


Figure 2.11: Processing of 4×4 residue block from *transformation* to restoring

2.2 | AV1

Review *AV1 Bitstream and Decoding Process*

*Development Process

*AOMedia companies

*Comparation with past generations

*Introduction of modules not present on other video codecs

*Block diagram

2.3 | Performance Analysis

*Compression gains

*Quality assessment

*Complexity (general/modules) and timing issues

References

[1] Rafael C. Gonzalez and Richard E. Woods. *Digital Image Processing*. eng. Fourth edition. OCLC: 987436552. New York, NY: Pearson, 2018. ISBN: 978-0-13-335672-4.

[2] Yun Qing Shi and Huifang Sun. *Image and Video Compression for Multimedia Engineering: Fundamentals, Algorithms, and Standards*. eng. 2. ed. Image Processing Series. OCLC: 254564955. Boca Raton, Fla.: CRC Press, 2008. ISBN: 978-0-8493-7364-0.

[3] *Anatomy of the Eye: Human Eye Anatomy*. en. https://owlcation.com/stem/Anatomy-of-the-Eye-Human-Eye-Anatomy.

[4] Kathy Mullen. "The Contrast Sensitivity of Human Color Vision to Red-Green and Blue-Yellow Chromatic Gratings". In: *The Journal of physiology* 359 (Mar. 1985), pp. 381–400.

[5] *Freepik — Download Now Millions of Free Vectors, Photos and PSD Files*. en. https://www.freepik.com.

[6] *YUV Sequences*. http://trace.eas.asu.edu/yuv/.

[7] Andrew Watson. "Efficiency of a Model Human Image Code". In: *Journal of the Optical Society of America. A, Optics and image science* 4 (Jan. 1988), pp. 2401–17.

[8] Kais Rouis and Chaker Larabi. "Perceptual Video Content Analysis and Application to HEVC Quantization Refinement". In: Nov. 2018, pp. 1–6.

[9] "AV1 Bitstream & Decoding Process Specification". en. In: (), p. 681.

CHAPTER 3

Video Coding Transforms

3.1 | Introduction

As mentioned previously, the basic principle behind the compression of video, is the reduction of inter-pixel/inter-symbol correlation. The various integral blocks of a video compression system try to accomplish this objective through different strategies. The *Intra-frame* and *Inter-frame Prediction* exploit spatial and temporal correlation, respectively. Through the subtraction of the input by the output of one of these blocks, and the attainment of the *residue*, the next compression stage is made in the *Transform* block, which is the focus of this work.

The technique implemented by this process relies on the energy compaction in the frequency domain to reduce the correlation within a frame block, i.e. the input of the Transform block is evaluated on its main frequencies — the *transform coefficients* — on a spatial domain, similarly to the process executed on a *Fourier Transform*. Once each block is quantized on these coefficients, the compression is made with the removal of the least significant ones, on the *Quantization* stage. The intent of the *transform* is to split the image into a set of predefined coefficients, that get transmitted in place of the *residue*.

The objective of this chapter is to give the reader a basic understanding of the theoretical basis behind said Transformations, as well as to introduce the most commonly used ones.

3.2 | Background

3.2.1 | Basis vector/image interpretation

A useful interpretation, and a good starting point to the study of this process, is to see it as the decomposition of the input as a set of basis vectors (1D transforms) or images/matrices (2D transforms). The transformation outputs , T_i, can be seen as the weights of each basis vector/image, $\vec{e_i}$, that summed return the input, $\vec{g}$, i.e.

$$\vec{g} = \sum_{i=1}^{N} T_i \vec{e_i} \tag{3.1}$$

which means that the coefficients are related to the amount of correlation between the input and each basis component, and can be obtained with the *inner product* of the input and each basis vector.

$$T_i = \vec{e_i}^T \vec{g} \tag{3.2}$$

Since each input vector will have different correlation values between the various basis vectors, this operation accomplishes two main objectives:

- De-correlation of the input values
- Signaling of the most important basis vectors.

Considering a 2D image, $g(x, y)$, and its corresponding transformed coefficients, $T(u, v)$, where (x, y) are the pixel coordinates, and (u, v) are the corresponding coordinates in the transform domain, we can obtain an analogous version of equation 3.2 as

$$T(u,v) = \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} g(x,y) f(x,y,u,v) \tag{3.3}$$

Similarly, we can re-obtain the original picture

$$g(x,y) = \sum_{u=0}^{M-1} \sum_{v=0}^{N-1} T(u,v) i(x,y,u,v) \tag{3.4}$$

where $f(x, y, u, v)$ and $i(x, y, u, v)$ are the *forward* and *inverse transformation kernels*. To better explain the concept of these, first it's needed to introduce the two following concepts.

3.2.1.1 | Separability

A useful characteristic of 2D Video Coding Transforms is its ability to be independently calculated between rows and columns. This means that given a 2D block as input, the transform coefficients can be calculated first with the *horizontal transform*, and then with the *vertical transform*, or vice-versa.

This aspect is applicable if the following conditions are applied

$$f(x,y,u,v) = f_1(x,u) f_2(y,v) \tag{3.5}$$

$$i(x,y,u,v) = i_1(x,u) i_2(y,v) \tag{3.6}$$

This means that the equation 3.3 is reconstructed as 2 independent and sequential operations

$$T_{temp}(x,v) = \sum_{y=0}^{N-1} g(x,y) f_2(y,v) \tag{3.7}$$

$$T(u,v) = \sum_{x=0}^{M-1} T_{temp}(x,v) f_1(x,u) \tag{3.8}$$

On AV1, due to the various implemented transformation kernels, this aspect is severely explored, since the only way of implementing the combination of different 1D kernels, is to calculate them independently. This aspect is further explained with the following concept.

3.2.1.2 | Symmetry

Taking equation 3.5, a transformation kernel is said to be symmetric if

$$f_1(y, v) = f_2(x, u) \tag{3.9}$$

This characteristic is particularly useful because it makes the forward and inverse transformations expressible as matrix multiplications. Therefore, the equations 3.3 and 3.4 are represented, respectively, as

$$T = F^T G F \tag{3.10}$$

$$G = I^T T I \tag{3.11}$$

where F and I are the forward and inverse transform matrices. This aspect is only possible for square matrix, i.e., input blocks with the same height and width.

This concept isn't exploited in AV1, since the use of different 1D transformation kernels, and rectangular block sizes ($M \neq N$) make the 2D transform asymmetric, and therefore, not executable as matrix multiplication. Consequently, the block transformation is made as 2 separate 1D operations, as shown previously.

Looking now at equation 3.4, we can interpret the inverse transformation kernel as a set of basis images, dependent of the (u, v) pair. By this, it is meant

$$g(x, y) = \sum_{u=0}^{M-1} \sum_{v=0}^{N-1} T(u, v) I_{u,v} \tag{3.12}$$

where

$$I_{u,v} = \begin{bmatrix} i(0,0,u,v) & i(0,1,u,v) & \dots & i(0,M-1,u,v) \\ i(1,0,u,v) & i(1,1,u,v) & \dots & i(1,M-1,u,v) \\ \vdots & \vdots & \dots & \vdots \\ i(N-1,0,u,v) & i(N-1,1,u,v) & \dots & i(N-1,M-1,u,v) \end{bmatrix} \tag{3.13}$$

Therefore, the forward and inverse transformation process can be seen as the deconstruction of an input block, into a set of $M \cdot N$ basis images, dependent of the used transformation kernel. As expressed in equations 3.5 and 3.6, this analogy can be made on a 1D space .

Given a general comprehension of the theoretical principles behind the *Transform* block, now the most common transformation kernels are introduced, with focus on the AV1 video codec.

3.3 | Transformation Kernels

3.3.1 | Discrete Fourier Transform (DFT)

Although it isn't implemented in video coding, it's widely used in digital signal processing, and many of the used transformation kernels are approximations of this function.

It has it's roots on the *Fourier Transform*, whose forward and inverse transformations are expressed in equations 3.14 and 3.15, respectively.

$$T(u,v) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} g(x,y)e^{-j2\pi(ux+vy)}dx\,dy \tag{3.14}$$

$$g(x,y) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} T(u,v)e^{j2\pi(ux+vy)}du\,dv \tag{3.15}$$

Once considered a finite number of points, the previous equations become

$$T(u,v) = \frac{1}{MN}\sum_{x=0}^{M-1}\sum_{y=0}^{N-1} g(x,y)e^{-j2\pi\left(\frac{ux}{M}+\frac{vy}{N}\right)} \tag{3.16}$$

$$g(x,y) = \sum_{u=0}^{M-1}\sum_{v=0}^{N-1} T(u,v)e^{j2\pi\left(\frac{ux}{M}+\frac{vy}{N}\right)} \tag{3.17}$$

which corresponds to replacing the kernels in equations 3.3 and 3.4 with

$$f(x,y,u,v) = \frac{1}{MN}e^{-j2\pi\left(\frac{ux}{M}+\frac{vy}{N}\right)} \tag{3.18}$$

$$i(x,y,u,v) = e^{j2\pi\left(\frac{ux}{M}+\frac{vy}{N}\right)} \tag{3.19}$$

The position of the multiplication factor, $\frac{1}{MN}$, is irrelevant, and in some works is divided into two terms in the forward and inverse kernels, $\frac{1}{M}$ and $\frac{1}{N}$, or even $\frac{1}{\sqrt{MN}}$.

Because of the use of complex numbers, this operation tends require a high computational effort, whence its disuse in video coding.

3.3.2 | Discrete Walsh-Hadamard Transform (WHT)

This transformation replaces the sum of sines and cosines of the DFT, alternating of positive and negative 1's, depending on the binary representation of the inputs.

Considering the inputs of the transform to be represented with m bits, where $m-1$ is the most significant bit (b_{m-1}), the forward and inverse kernels are represented as

$$f(x,y,u,v) = i(x,y,u,v) = \frac{1}{\sqrt{MN}}(-1)^{\sum_{i=0}^{m-1}\lfloor b_i(x)p_i(u)+b_i(y)p_i(v)\rfloor} \tag{3.20}$$

where

$$\begin{aligned} p_0(u) &= b_{m-1}(u)\\ p_1(u) &= b_{m-1}(u) + b_{m-2}(u)\\ &\vdots\\ p_{m-1}(u) &= b_1(u) + b_0(u) \end{aligned} \tag{3.21}$$

3.3.3 | Discrete Cosine Transform (DCT)

The most commonly used transform, the *DCT*, was published by Ahmed et al. in 1974 [3]. Since then, it has been adopted in a wide range of applications, being the only transform used in the first generations of video codecs, as well as in *still image compression*, being the basis of the *JPEG* standard.

It is frequently compared to the *DFT*, due to the similarity of their operation. However, as the name implies, the *DCT* relies on the cosine function to create its basis images, which is a *periodic* and *symmetrically even* function. Therefore, as mentioned by [4, A. V. Oppenheim], *"Just as the DFT involves an implicit assumption of periodicity, the DCT involves implicit assumptions of both periodicity and even symmetry"*. This is easily observable once considered the equivalent process of both algorithms. Taking an N-point sequence, $g(n)$, the calculation of the *DFT* and *DCT* of such sequence is equivalent to the processes presented at table 3.1.

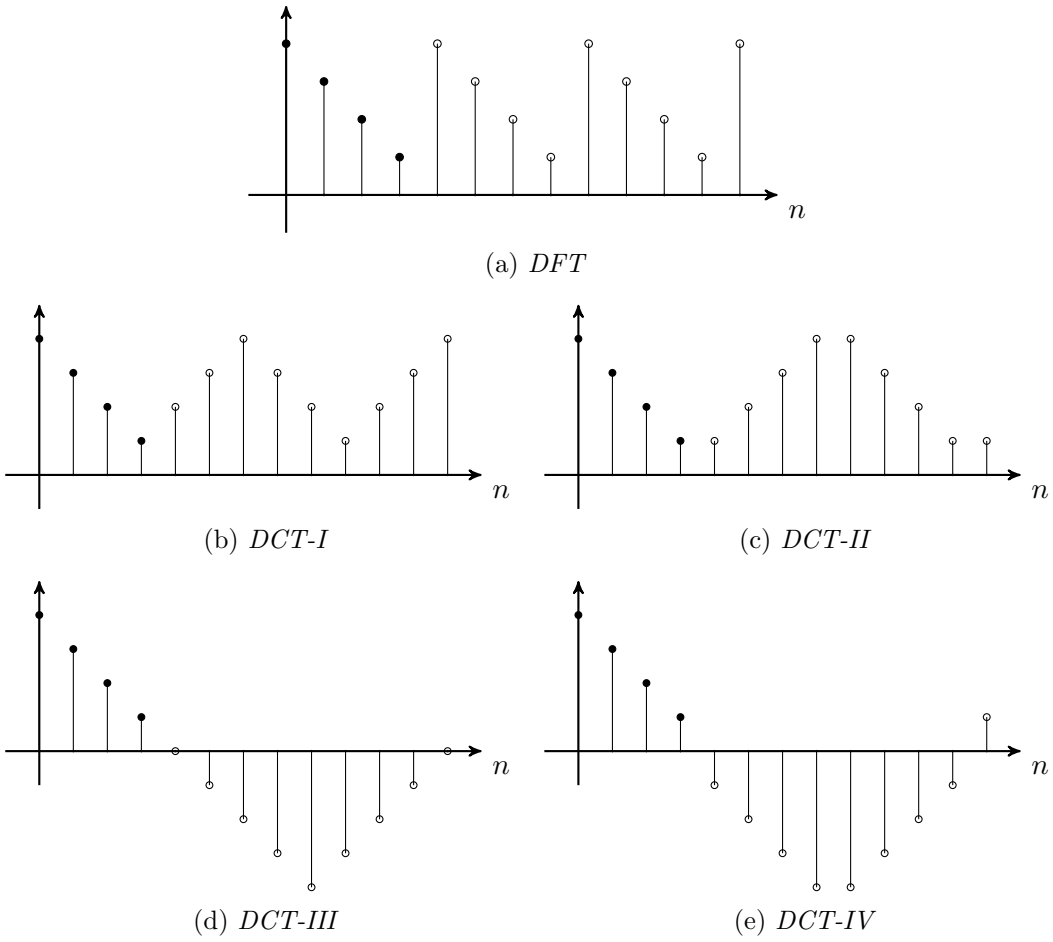
Step	***DFT***	***DCT***
1	Repeat $g(n)$ every N points, giving origin to $\widetilde{g}_N(n)$	Concatenate $g(n)$ with a flipped version of itself, creating a $2N$ sequence, $g_{2N}(n)$, and repeat it, giving origin to $\widetilde{g}_{2N}(n)$
2	Calculate the *Fourier* expansion of $\widetilde{g}_N$	Calculate the *Fourier* expansion of $\widetilde{g}_{2N}$
3	Keep the first N coefficients and set all others to 0	Keep the first N coefficients, and set all others to 0

Table 3.1: Similarity between the processes of the *DFT* and the *DCT*

The main reason behind the heavy adoption of the *DCT* is its great energy compaction on the lower frequencies, where most of the energy in a picture is packed. If the output of the first step of table 3.1 is observed, this aspect is more easily understood. In figure 3.1, a 4 point sequence, corresponding to the filled points, gets replicated throughout the discrete time axis, according to the corresponding transform.

Due to the back-to-head repetition seen in figure 3.1a, there is a disruption every N points, which gives origin to high frequency components in the *Fourier* transform. Therefore, the more continuous behavior obtained with the back-to-back repetition of the *DCT* gives origin to more significant low frequency coefficients. However, there are many ways of creating a periodic and symmetric sequence from a finite number of points. This factor has led to the implementation of different versions of the *DCT*, which differ in minor details between themselves. These differences are consequence of the way the symmetry is obtained, which can be observed in figures 3.1b to 3.1e. The represented implementations are referred to as *DCT-I* to *DCT-IV*, but other possibilities exist. Their definition depends on the overlapping of points when repeating each sequence.

Since the *DCT* in *AV1* is implemented in one dimension, the description of the following kernels is also made in 1D. Therefore, the dimension of the transform, L, is referring either to the blocks' width or height, depending if the operation is made to the rows or columns, respectively (M or N, previously).



(a) *DFT*

(b) *DCT-I*

(c) *DCT-II*

(d) *DCT-III*

(e) *DCT-IV*

Figure 3.1: Sequences generated in the first step of table 3.1for the DFT and different DCTs. Filled dots correspond to the original sequence.

DCT-I The sequence created with first version of the DCT has overlapping points at $n = k(L-1),\ k = 0, 1, 2, ...$, making the overall period of the final sequence $2L - 2$.

$$f(x,u) = \frac{2}{L-1}\alpha(x)\cos\left(\frac{\pi x u}{L-1}\right) \tag{3.22}$$

where

$$\alpha(x) = \begin{cases} \frac{1}{2}, & x = 0 \vee x = N-1 \\ 1, & 1 \leq x \leq N-2 \end{cases} \tag{3.23}$$

The inverse transform becomes

$$i(x,u) = \alpha(u)\cos\left(\frac{xu\pi}{L-1}\right) \tag{3.24}$$

DCT-II Usually referred to as "the *DCT*", it is by far the most implemented version, being the only one mentioned in many of the studied works.

As seen in figure 3.1c, this version has no overlap on the created sequence, making the period $2L$, and the points of symmetry $kL - \frac{1}{2}$.

$$f(x,u) = i(x,u) = \beta(u)\cos\left(\frac{(2x+1)u\pi}{2L}\right) \tag{3.25}$$

$$\beta(u) = \begin{cases} \sqrt{\frac{1}{L}}, & u = 0 \\ \sqrt{\frac{2}{L}}, & 1 \leq u \leq N-1 \end{cases} \tag{3.26}$$

DCT-III Named the *inverse* of DCT-II, due to the switch of the transform and pixel coordinates.

$$f(x,u) = i(x,u) = \beta(u)\cos\left(\frac{(2u+1)x\pi}{2L}\right) \tag{3.27}$$

$$\beta(u) = \begin{cases} \sqrt{\frac{1}{L}}, & u = 0 \\ \sqrt{\frac{2}{L}}, & 1 \leq u \leq N-1 \end{cases} \tag{3.28}$$

DCT-IV Is the basis of the *Modified Discrete Cosine Function (MDCT)*, where the input blocks overlap.

$$f(x,u) = i(x,u) = \sqrt{\frac{2}{L}}\cos\left(\frac{(2u+1)(2x+1)\pi}{4L}\right) \tag{3.29}$$

3.3.4 | Discrete Sine Transform (DST)

Similarly to the DCT, there is also the possibility to represent a finite sequence as a sum of discrete *sine* functions, giving origin to the *DST*. Contrarily to the former presented transform, this variant uses sinusoidal functions to generate its basis images, which gives origin to *odd symmetric* sequences.

In the same way as its *even* counterpart, there are various different ways off accomplishing such symmetry, which also gives origin to eight different variations of this Transform. However, due to its misuse over the DCT, only the *DST-II* is presented.

$$f(x,u) = i(x,u) = \sqrt{\frac{2}{L+1}}\sin\left(\frac{(j+1)(u+1)\pi}{L+1}\right) \tag{3.30}$$

Equivalently to what happens with the DFT, the odd symmetry of this function gives origin to discontinuities, which are undesirable when coding video blocks, since they lead to less significant low frequency coefficients, and therefore higher quantization errors.

3.3.5 | Asymmetric Discrete Sine Transform (ADST)

The symmetric behavior of previous transforms lead to better performance on evenly spread residue blocks, i.e. when the pixel values post-subtraction (and before transformation) have roughly the same value across the whole block.

However, due to the directional spatial prediction, the residue on one boundary of the block may be different from the other boundaries, since the chosen direction for prediction may diverge from the original block, across the prediction block. This leads to worse energy compression.

In order to combat this aspect, VP9 introduced a new transform called *Asymmetric Discrete Sine Transform (ADST)*, which corresponds to an alternative implementation of the DST with the addition of frequency and phase shifts.

This enhancement provides the developer with a high degree of liberty, since the basis images can be adapted with the variation of the shifts. On AV1, there is only one ADST implementation per block size. However this transformation can be done in two directions, i. e., the input vector can be transformed front-to-back and vice-versa. *AOMedia* named these transforms *ADST* and *Flip-ADST*, according to the direction of the input vector.

3.4 | Integer Transformations

In battery driven applications, computing power plays an important role. Consequently, any approach that leads to lower computational costs tends to get incorporated into a video codec.

When considering the transformation stage, a widely used approach is the use of *integer transforms*. The objective of such operations is to maintain the features of floating point transforms, but severely reducing the complexity, decreasing the used operations to arithmetic additions and integer multiplications. In many implementations, the latter are implemented with bitwise shifts and additions.

From the transforms presented throughout section 3.3, there have been several methods of developing integer counterparts. Most of the fast implementations are based in either *Fast Fourier Transform* algorithms or in the *Walsh-Hadamard Transform* [10, 11]. Since the objective of this work was to develop a *Transform Co-processor* for *libaom*, the focus of this section resolves around these kernels.

Therefore, the reference software was profoundly studied, especially regarding the transformation stage, which is represented in figure 3.2.

This stage is controlled by a configuration set, which is chosen according to the desired encoding objectives. These parameters control the transformation block width and height (`size_col` and `size_row` [1], respectively), the transformation kernels to use in the rows and columns, the precision to use in the sine and/or cosine coefficient approximations, as well as other parameters for overflow control. Associated to the transformation kernel chosen, the variables `ud_flip` and `lr_flip` are also set. The first one is set to `1` if the block's columns are to be transformed with the *Flip-ADST* kernel. If such choice is applied to the rows, the second variable is, likewise, set to `1`. These variables control if the input rows are flipped vertically, and/or if the coefficients resulting from the column transformation are flipped horizontally [2].

The choosing of these parameters will not be addressed in this work, since *AV1* allows for a great deal of maneuverability to the designer, as to adjust each encoder/decoder to the desired application. In this regard, *libaom* allows for a high number of configuration options, that dramatically change the parameters chosen in the transformation stage, as well as in the rest of the system.

Throughout the represented process, many of the operations are done with sequential, iterative processes, e.g., the input vector selection or the flipping operations. Such operations

[1] Each number does not correspond to the number of elements in columns and rows, but rather to the number of rows and columns.

[2] Here, the notation of *horizontally* or *vertically* is set considering a matrix input block. In the 1D transform implemented in *libaom*, this just means that what would be the last coefficient is now the first, and so on.

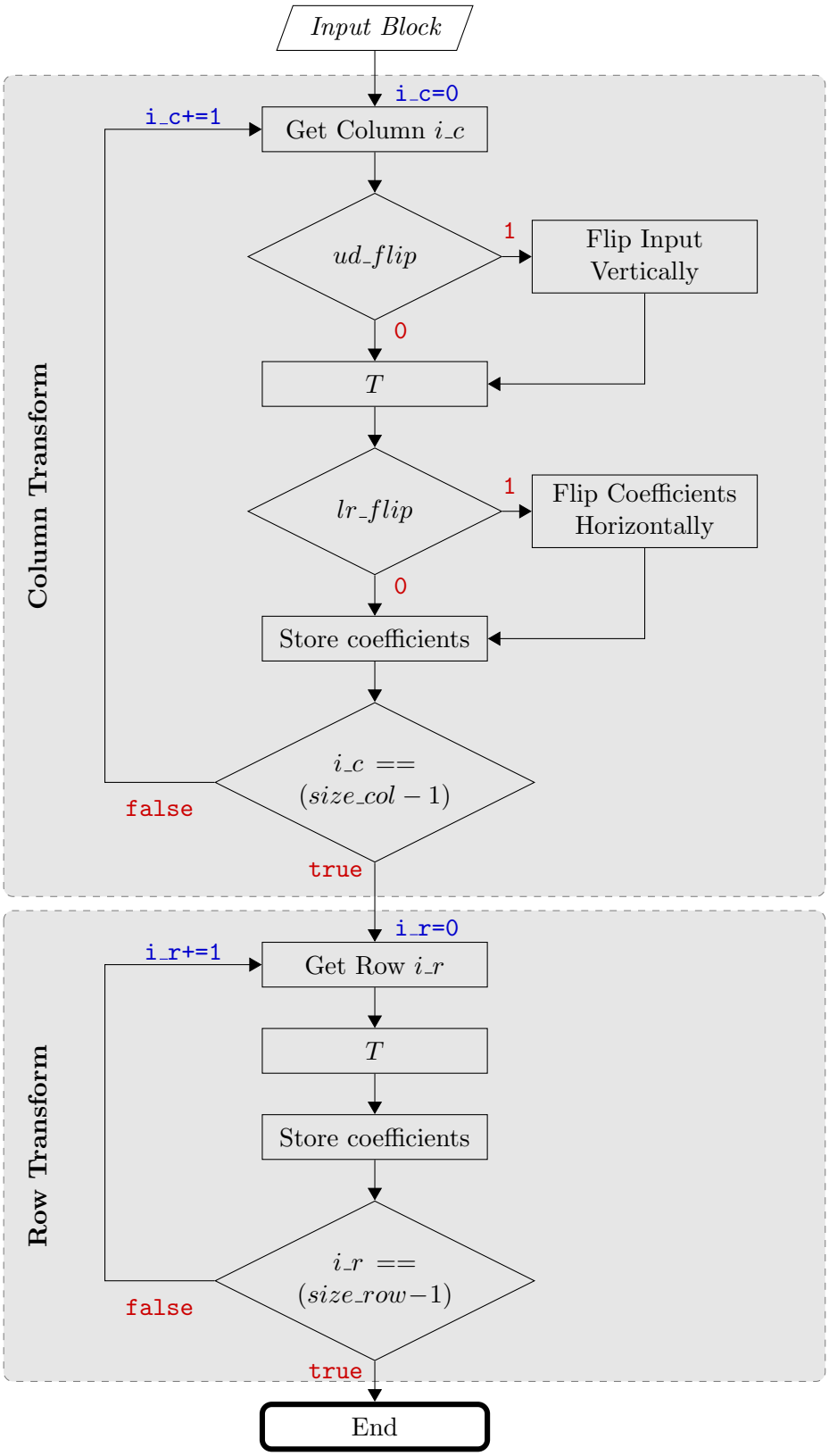


Figure 3.2: Flowchart of the Transform Stage on *libaom*

would greatly benefit of a hardware implementation, since they are easily parallelizable, as the objectives of *AV1* suggested. However, on this work, the focus relies of the *T* block, i.e., the transformation itself.

Independently of the transformation kernel, the operation is done sequently, in various stages. In each of these, the corresponding intermediary coefficients get calculated as function of two of the previous calculated coefficients. These, in most of the stages, are multiplied by a specific integer approximation of cosine/sine value. Such approximations, as mentioned previously, depend on the number of bits chosen on which they are represented. The arrays on which the calculated cosine and sine values are stored have 64 and 9 positions, respectively, representing the first quadrant of the trigonometric circle. Each position is calculated according to equations 3.31 and 3.32, where k represents the position in the array, and N corresponds to the number of bits. Therefore, `cospi[0]` corresponds to $\cos(0)$, `cospi[63]` is $\cos(63\pi/128)$, and the following positions can also be associated to a certain angle.

$$cospi[k] = \left\lfloor 2^N \cos\left(\frac{k\pi}{128}\right)\right\rceil \tag{3.31}$$

$$sinpi[k] = 2^N \left\lfloor \frac{2}{3}\sqrt{2}\sin\left(\frac{k\pi}{9}\right)\right\rceil \tag{3.32}$$

The *sinpi* array is only used in the shortest length of the *ADST*, which is the reason it only has five positions. All other versions of this kernel use *cospi* to get the corresponding value.

Most of the temporary coefficients inside each stage are calculated with the function `half_btf`, which performs the operation represented in equation 3.33. This function takes the two previously calculated coefficients, two values from the previously introduced arrays, as well as the number of bits used to represent these, and maps the result from the multiplications and sum of the first inputs to the desired number of bits.

$$\texttt{half\_btf(w0, in0, w1, in1, bits)} \;\widehat{=}\; \left\lfloor \frac{w_0 in_0 + w_1 in_1 + 2^{bits-1}}{2^{bits}} \right\rfloor \tag{3.33}$$

Although the code implementation is sequential, the *8 length* transformation kernels are represented in figures 3.4 and 3.6 as parallel block diagrams, with the diverse stages in series. While *AV1* accepts *transform block* sizes varying between 4 and 64, the method of transformation is similar between the different sizes.

Both pictures start with the input vector components, i.e., `x0` to `x7`. The following sum's represent the addition of the two pointing values, in case the that the arrow guiding these doesn't present any further coefficient. If such is verified, the operation to be realized is the one presented in equation 3.33. The value near each arrow is referred to the equivalent `cospi` position, that multiplies by the result coming from the arrow's origin. Figure 3.3 presents a visual aid for the following figures.

The identity transforms, *IDTX*, are the simplest of the ones implemented in *libaom*, since they consist of a scale factor, which varies throughout the transform sizes.

On the 4 and 16 length transforms, the scaling factor includes a 12-bit integer approximation of the square root of 2, which is calculated through

$$N_{\sqrt{2}} = \left\lfloor 2^{12}\sqrt{2}\right\rceil = 5793 \tag{3.34}$$

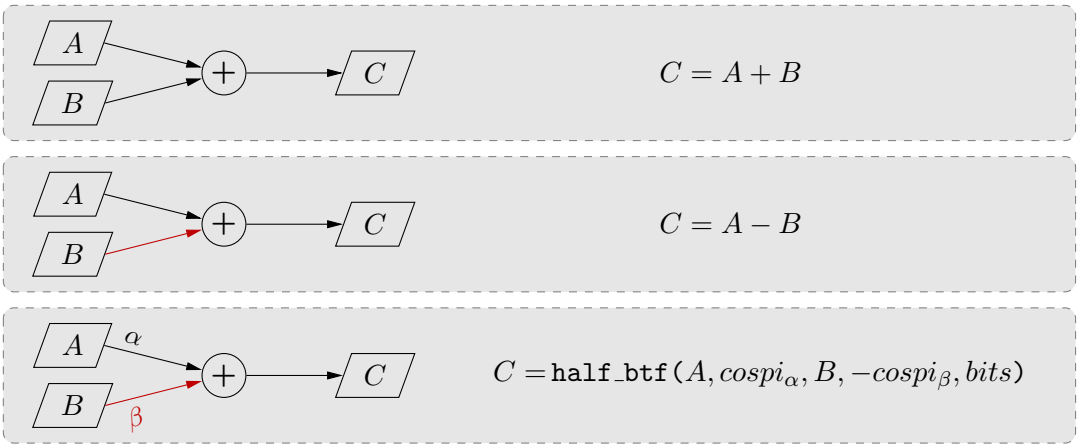


Figure 3.3: Graphical aid for figures 3.4 and 3.6

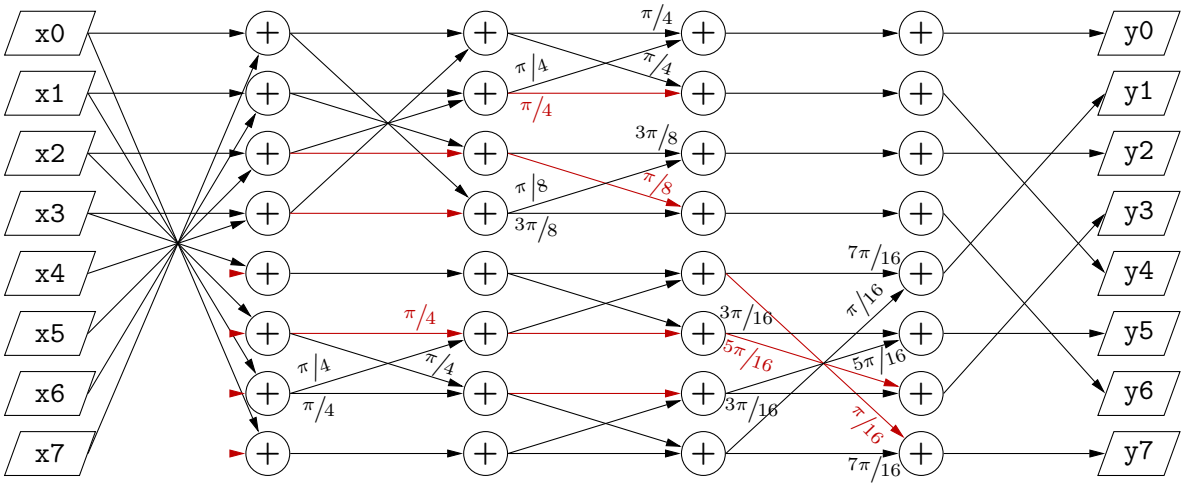


Figure 3.4: Block diagram of *libaom*'s Integer DCT

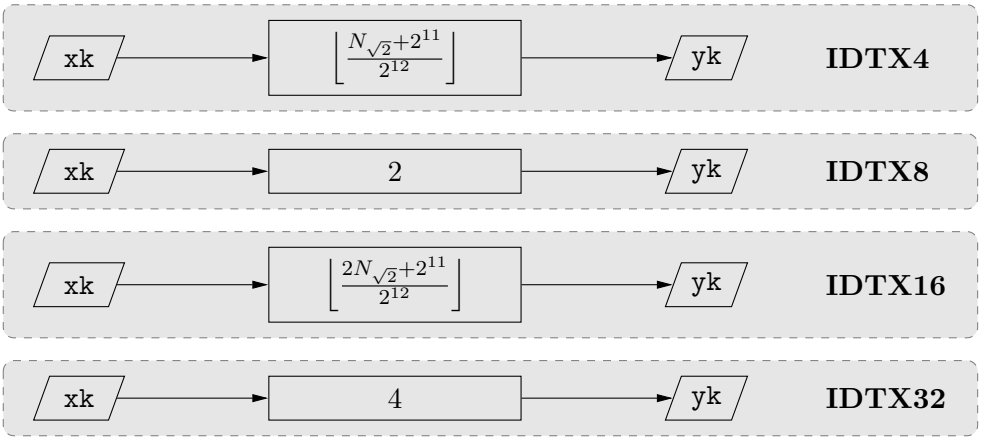


Figure 3.5: Description of the Identity transforms in *libaom*

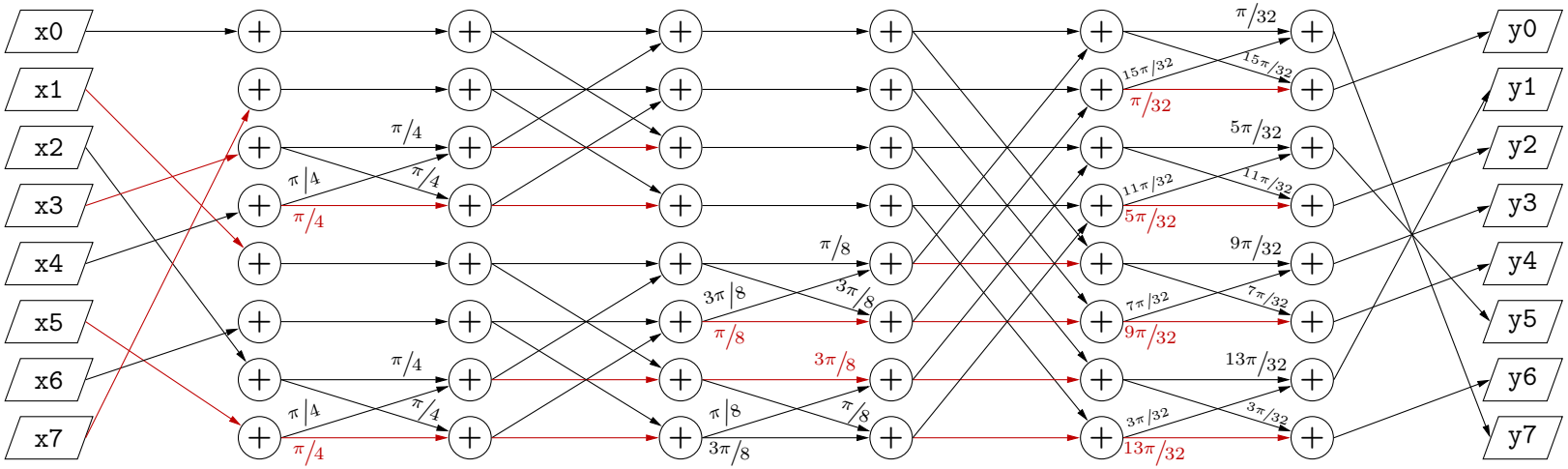


Figure 3.6: Block diagram of *libaom*'s Integer ADST

Being so, the input also suffers an additional mapping, similar to the operation in 3.33. These operations are demonstrated in figure 3.5.

With the forward transformations explained and represented graphically, it's easily understandable that the corresponding inverses correspond to the backwards operation in figures 3.4, 3.5 and 3.6. With this, it is meant that only the direction of the arrows change, and the corresponding procedure is done right-to-left, i.e., the output's position, `y`, is now the input.

References

[1] Yun Qing Shi and Huifang Sun. *Image and Video Compression for Multimedia Engineering: Fundamentals, Algorithms, and Standards*. eng. 2. ed. Image Processing Series. OCLC: 254564955. Boca Raton, Fla.: CRC Press, 2008. ISBN: 978-0-8493-7364-0.

[2] Rafael C. Gonzalez and Richard E. Woods. *Digital Image Processing*. eng. Fourth edition. OCLC: 987436552. New York, NY: Pearson, 2018. ISBN: 978-0-13-335672-4.

[3] N. Ahmed, T. Natarajan, and K. R. Rao. "Discrete Cosine Transform". In: *IEEE Transactions on Computers* C-23.1 (Jan. 1974), pp. 90–93. ISSN: 0018-9340.

[4] Alan V. Oppenheim, Ronald W. Schafer, and John R. Buck. *Discrete-Time Signal Processing*. eng. 2nd ed. Upper Saddler River, NJ: Prentice Hall, 1998. ISBN: 978-0-13-754920-7.

[5] *Discrete Cosine Transform - MATLAB Dct*.

[6] William K. Pratt. *Digital Image Processing: PIKS Inside*. eng. 3. ed. A Wiley-Interscience Publication. OCLC: 248389925. New York: Wiley, 2001. ISBN: 978-0-471-37407-7.

[7] Jingning Han, Yaowu Xu, and Debargha Mukherjee. "A Butterfly Structured Design of the Hybrid Transform Coding Scheme". en. In: *2013 Picture Coding Symposium (PCS)*. San Jose, CA, USA: IEEE, Dec. 2013, pp. 17–20. ISBN: 978-1-4799-0294-1 978-1-4799-0292-7.

[8] Benny Bing. *Next-Generation Video Coding and Streaming*. Hoboken: Wiley, 2015. ISBN: 978-1-119-13332-2 978-1-119-13333-9.

[9] Soo-Chang Pei and Jian-Juin Ding. "The Integer Transforms Analogous to Discrete Trigonometric Transforms". In: *IEEE Transactions on Signal Processing* 48.12 (Dec. 2000), pp. 3345–3364. ISSN: 1053-587X.

[10] S. Wolter et al. "Parallel Architectures for 8*8 Discrete Cosine Transforms". In: *[Proceedings] 1992 IEEE International Symposium on Circuits and Systems*. Vol. 1. San Diego, CA, USA: IEEE, 1992, pp. 149–152. ISBN: 978-0-7803-0593-9.

[11] Yonghong Zeng et al. "Integer DCTs and Fast Algorithms". In: *IEEE Transactions on Signal Processing* 49.11 (Nov./2001), pp. 2774–2782. ISSN: 1053587X.

CHAPTER 4

Developed Architecture

4.1 | REEEEEEEEEEEEEEEEEEEEEEEEE

Sed commodo posuere pede. Mauris ut est. Ut quis purus. Sed ac odio. Sed vehicula hendrerit sem. Duis non odio. Morbi ut dui. Sed accumsan risus eget odio. In hac habitasse platea dictumst. Pellentesque non elit. Fusce sed justo eu urna porta tincidunt. Mauris felis odio, sollicitudin sed, volutpat a, ornare ac, erat. Morbi quis dolor. Donec pellentesque, erat ac sagittis semper, nunc dui lobortis purus, quis congue purus metus ultricies tellus. Proin et quam. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos hymenaeos. Praesent sapien turpis, fermentum vel, eleifend faucibus, vehicula eu, lacus.