Linelet, an Ultra-Low Complexity, Ultra-Low Latency Video Codec for Adaptation of HD-SDI to Ethernet

Hans-Kristian Arntzen

Electronics System Design and Innovation
Submission date: June 2014
Supervisor: Andrew Perkis, IET
Co-supervisor: Odd Inge Hillestad, Nevion

Norwegian University of Science and Technology
Department of Electronics and Telecommunications

Problem description

Line-based video compression for next generation broadcast signals over IP

In todays broadcast facilities, real-time digital video and associated audio and metadata is transported within the facility using HD-SDI, a serial digital interface on an coax cable or optical fiber. A very recent trend is that broadcasters are looking into how they can save cost and improve workflow by moving towards transporting SDI signals over a packet-based commodity Ethernet networking infrastructure, using e.g. SMPTE 2022-6 for SDI over IP. [1]

The bitrate of the HD-SDI payload alone is approximately 1.5 Gb/s, which means that video edge devices and network access equipment need to be designed for 10GE. However, at the same time, a new generation of production image formats are emerging, such as 4K/8K or Ultra HDTV (UHD). With higher resolution and potentially higher frame rate, the bitrate of the corresponding SDI signals increase significantly. For example, 4K/UHD-1 at 50 frames/sec with todays 4:2:2 10-bit YCbCr video sampling require a 12 Gb/s SDI signal. In order to take advantage of 10GE networking becoming commodity, some form of light-weight and near-zero latency video compression would be advantageous in these systems.

The project shall look into the requirements for appropriate video compression technologies [2] for such signals and develop, implement and evaluate a suitable algorithm.

[1]This initiative comes from the Joint Task Force on Networked Media, a collaboration between EBU, SMPTE and Video Services Forum. (https://tech.ebu.ch/Jahia/site/tech/cache/offonce/groups/jtnm) A call for technology was conducted in 2013 (https://tech.ebu.ch/docs/groups/jtnm/JT-NM_RFT-120913.pdf), and a report was published with the received proposals (https://tech.ebu.ch/webdav/site/tech/shared/groups/jtnm/GapAnalysisReport_231213.pdf).

[2]A current proposal sent to Joint Task Force on Networked Media is the codec ”TICO” from intoPIX (http://www.videoservicesforum.org/download/jtnm/JTNM012-1.zip). It is based on compressing single scanlines, with visually lossless compression from 1:2 to 1:4 compression rates.

Summary

In this project we have designed, specified, implemented, optimized and evaluated a new ultra-low latency ($\leq$ 1 ms) and ultra-low complexity intra-only video compression codec, *Linelet*, which is able to compress 1080p50 and beyond in real-time on regular desktop PC equipment. There are strong indications that 4K@60 encoding is possible in real-time on powerful desktop equipment.

The broadcasting industry today is moving towards 4K and ultra-high definition resolutions and this puts greater burdens on transmission equipment which need to transmit the highest quality video material during production in real-time. We therefore see the need for a lightweight compression solution which can keep the bandwidth down while keeping the pristine quality needed for production. Such a solution must be cheap, fast and have near-zero latency in order to justify the cost of adding compression. With a lightweight compression scheme we enable the possibility to transmit production video over Ethernet links. We can compress 1.5 Gbit/s and 3.0 Gbit/s HD video into a 1 gigabit ethernet link, or 12 Gbit/s 4K video down to a 10 gigabit ethernet link.

Linelet is based on the discrete wavelet transform and uses the 5/3 wavelet filter for simplicity and reversibility of all operations. Linelets focus is transforming video data horizontally for near-zero latency, but also allows for a simple method to exploit vertical redundancies. Exploiting vertical redundancies lead to a 3 dB improvement in PSNR and vastly improved visual quality over a design which only considers horizontal redundancy.

Linelets target use case is to be implemented in either an FPGA or ASIC, working with uncompressed video signals in real-time. The codec is designed ground-up for simplicity, requires little memory and avoids any expensive arithmetic. Entropy coding is vastly simplified over conventional approaches which ensures very high encoder throughput (5 Gbit/s and beyond) even on desktop PC hardware.

Using our software implementation of Linelet, we performed a small-scale subjective evaluation with experts in the field based on recommendations in ITU-R BT.500 and the evaluation indicates that Linelet remains visually lossless at 1:2, 1:4 and even 1:6 compression rates for very difficult test sequences such as ParkJoy and Horse at a viewing distance of 3H. The experts were very familiar with these sequences. Further tests should be carried out to verify this.

Sammendrag (Norwegian)

I dette prosjektet har vi designet, spesifisert, implementert, optimisert og evaluert en ny ultra-low latency ($\leq$ 1 ms) og ultra-low complexity intra-only videokompresjonskodek, *Linelet*, som kan komprimere 1080p50 i sanntid på en vanlig stasjonær PC. Det er sterke indikasjoner på at 4K@60 vil være mulig i sanntid på en kraftig stasjonær PC.

Kringkastingsbransjen ser i dag på 4K og ultra-høydefinisjonsformater og dette medfører enda større belastning for transmisjonsutstyr som trenger å øverføre den høyeste kvalitet i produksjon i sanntid. Vi ser derfor nytten i en lettvekts komprimeringsløsning som kan holde båndbredden nede samtidig som vi beholder den ypperste kvalitet som trengs i produksjon. Løsningen må være billig, rask og ha et tilnærmet ikke-eksisterende etterslep for at kompresjon skal være nyttig. Med lettvekts-kompresjon legger vi til rette for at produksjonsvideo kan sendes over IP nettverk. Vi kan komprimere 1.5 Gbit/s og 3.0 Gbit/s HD ned til en 1 gigabit Ethernet-link eller 12 Gbit/s 4K video ned til en 10 gigabit Ethernet-link.

Linelet er basert på en diskret wavelet-transform og bruker 5/3-filteret for enkelhet og reversibilitet i alle ledd. Linelet fokuserer på å transformere video horisontalt for tilnærmet ingen etterslep, men tillater også en enkel metode for å utnytte redundans vertikalt. Å utnytte vertikal redundans førte til en forbedring på 3 dB PSNR og store forbedringer visuelt over en løsning som bare tok hensyn til horisontal redundans.

Linelets bruksområde er i enten en FPGA eller ASIC der den kan jobbe med ukomprimerte videosignaler i sanntid. Kodeken er designet fra bunnen av med tanke på enkelhet, krever lite minne og unngår dyre kostbare operasjoner. Entropikodingen er svært redusert i kompleksitet i forhold til konvensjonelle metoder, noe som muliggjør svært høy ytelse (5 Gbit/s og mer) selv med en stasjonær PC.

Med software-implementasjonen vår utførte vi en småskala subjektiv evaluering med eksperter fra bransjen basert på retningslinjer i ITU-R BT.500. Evalueringen indikerer at Linelet holder seg visuelt tapsfri på 1:2, 1:4 og selv 1:6 kompresjonsrater for veldig vanskelig testmateriale som ParkJoy og Horse på en avstand tre ganger skjermhøyde. Ekspertene var veldig godt kjente med disse test-sekvensene. Flere tester bør utføres for å verifisere testresultatene.

Contents

List of Figures

List of Tables

Abbreviations

3D-TV	=	3D television
4K	=	resolution with approximately 4000 horizontal pixels
8K	=	resolution with approximately 8000 horizontal pixels
API	=	application programming interface
ASIC	=	application specific integrated circuit
AVC	=	advanced video coding
BBC	=	British Broadcasting Corporation
CABAC	=	context adaptive binary arithmetic coding
CGI	=	computed generated imagery
CPU	=	central processing unit
CRT	=	cathode ray tube
CSF	=	contrast sensitivity function
DC	=	direct current (0 Hz)
DCI	=	digital cinema initiative
DCT	=	discrete cosine transform
DFT	=	discrete fourier transform
DPCM	=	differential pulse code modulation
DSIS	=	double-stimulus impairment scale
DWT	=	discrete wavelet transform
EBCOT	=	embedded block coding with optimal truncation
EBU	=	european broadcasting union
EZW	=	embedded zero-tree wavelets
FPGA	=	field-programmable gate array
GOP	=	group of pictures (MPEG)
GPGPU	=	general-purpose graphics processing unit
HD	=	high-definition
HDMI	=	high-definition multimedia interface
HD-SDI	=	high-definition serial digital interface
HDTV	=	high-definition television
HEVC	=	high-efficiency video coding
HVS	=	human visual system
IP	=	internet protocol
ISO	=	international organization for standardization
ITU	=	international telecommunication union
JPEG	=	joint photographic expert group
JT-NM	=	joint task force on networked media
KLT	=	Karhunen-Loeve transform
LCD	=	liquid crystal display
LSB	=	least significant bit (or byte)
LUT	=	look-up table
MDCT	=	modified discrete cosine transform
MOS	=	mean opinion score
MPEG	=	moving picture experts group
MSB	=	most significant bit (or byte)

MSE	=	mean square error
NHK	=	Nihon Hoso Kyokai
NTSC	=	national television system committee
PAL	=	phase-alternating line
PC	=	personal computer
PSNR	=	peak signal-to-noise ratio
RCT	=	reversible color transform
RGB	=	red-green-blue
SDI	=	serial digital interface
SD-SDI	=	standard-definition serial digital interface
SDTV	=	standard-definition television
SECAM	=	séquentiel couleur à mémoire
SIMD	=	single-instruction, multiple data
SMPTE	=	society of motion picture and television engineers
SPIHT	=	set partitioning in hierarchical trees
SSE	=	streaming SIMD extensions
SSIM	=	structural similarity
TV	=	television
UHD	=	ultra-high definition

Chapter 1

Introduction

1.1 Broadcasting

Broadcasting is the distribution of video and audio to a large audience simultaneously. The communication is one-way, from the broadcaster to multiple receivers. It is of great commercial and public interest to provide television broadcasting to the population at large. Television broadcasting today is a multi-billion dollar industry with consumers in nearly every household of the developed world.

When a consumer sees broadcast video content on a screen, the material has gone through several steps before it ever reaches the eyes of the viewer. Each step on the way represents opportunities for revenue.

- Content creation
- Contribution
- Content management and publishing
- Content distribution
- Display and reproduction technology
- Consumer

Innovations within this value chain are driven by constant desires from the industry to sell products and consumers who desire new content and improved quality. Ultimately, all value flows backwards from consumers. Without consumers, the value chain becomes meaningless. There is no need to shoot productions which are never seen.

Not only quality of the content itself, but the quality of how the content is reproduced can be just as important for the overall enjoyment of broadcast content. Display resolution is a concept that is often associated with quality. Higher resolutions allow smaller details to be present in the image, and sharp objects can remain sharp and crisp. Higher resolutions allow a more real world-like representation of content. Over the history of television broadcasting, screen resolutions have increased to meet consumer expectations of quality, and the resolution is still on the rise.

It is clear that at some point, increasing resolution further becomes meaningless. There is only so much detail we can hope to perceive as humans. Is our current HDTV good enough? Is there any real demand for higher resolutions? Will ultra-high definitions (UHD) allow us to experience content in a way that is not possible with the current high-definition television?

Before the arrival of digital television, analog broadcasting was the norm with standards such as NTSC, PAL and SECAM. With the arrival of digital television, digital equivalents of these standards were created with 576 video lines (576i) for PAL/SECAM and 480 lines (480i) for NTSC. This family of formats is referred to as standard-definition television (SDTV).

After a couple of decades experimenting with higher resolution broadcasting, standards bodies settled on our current definition of high-definition television (HDTV). It is a format with either 720 lines progressive (720p), 1080 lines interlaced (1080i) or 1080 lines progressive (1080p) as defined by ITU-R BT.709 [6]. The 1080 lines format is often referred to as ”full-HD”.

The industry believes ultra-high definition resolutions are meaningful, and they are starting to look at 4K and UHD video formats which go far beyond the current standard resolutions for HDTV. The promise is unparalleled picture quality and the sense of ”being there”.

For broadcasting of UHD, two formats have been standardized in ITU-R BT.2020 [7]. These are 3840x2160 and 7680x4320 resolutions. Going from the 1920x1080 format (1080p), these new formats are exactly 2 and 4 times the number of pixels horizontally and vertically. NHK of Japan have committed themselves to 8K technology (Super Hi-Vision) [8], pushing the boundaries of human perception even more. Their goal is to capture and transmit the 2020 Tokyo Olympics in this format. They have conducted experimental transmissions over the air with this format successfully.

Right now, 4K is in a technology push stage. The industry is hoping for 4K to be a commercial success, but there is not yet a convincing argument for consumers to go out and buy 4K TV sets. There is an obvious lack of content in 4K, which also was the case with the now-dead stereoscopic 3D trend. 4K might get critical mass with upcoming FIFA World Cup 2014 and 2016 Olympics which will likely push 4K material. High quality content for new display technology does not guarantee success however, as was the case with London 2012 Olympics failing to push 3D-TV technology. [9]

Stereoscopic display technology has been tried several times in the past, starting in the cinemas in the 1950s with red/cyan anaglyph glasses. The basis of current stereoscopic technology is to display two separate images to the eyes, and let the human brain recreate the 3D representation from that. This is not without problems however. There are several distortions which can occur when one tries to project a 3D space onto two screens, and then attempt to display that to the viewer. Many people are affected by headaches and severe eye-strain using this technology. The trend of 3D-TV sets in late 2000s and early 2010s faded out quickly, with CES 2013 marking the end of 3D-TV as the ”next big thing”. [10] 4K has taken over as the new trend. Simply increasing resolution is an easy way to progress forward as it can only improve current attributes (resolution), and not drastically change how things work. It is obvious that display manufactures are trying to create a demand for new TV sets after the 3D-TV flop. If everyone is happy with what they already have and unwilling to purchase new TV sets, TV manufacturers will eventually go out of business.

The perceived quality of a video image is not based on screen resolution alone. Some experts argue that for a jump to 4K to have any effect on visual quality, we cannot look at one parameter at a time, we must also improve the frame rate and color representation, possibly an high-dynamic range (HDR) representation. Doubling frame rate as well as jumping to 4K would not just quadruple required bandwidth, but require eight times the bandwidth instead, a dramatic increase which might not even be perceptible to the average viewer.

With higher and higher resolutions, for the quality improvement to even be visible, we need larger display devices. We also need to sit close enough to the screen such that a larger field of view can be covered. At some point, increasing the resolution becomes meaningless unless one is determined to sit closer to the screen or buy much larger TVs. If one sits too close, it becomes more and more difficult to have a good overview of the whole screen.

1.2 Technical challenges with 4K/UHD video

In order for 4K to become mainstream, technical challenges in the entire multimedia chain from production to distribution must be solved.

1.2.1 Codec technology

Distribution

For distribution to consumers, the new HEVC/H.265 [11] codec was released and standardized by a joint effort between ISO and ITU-T. Their previous video codec standard, H.264/AVC [12], has been an extremely successful standard to date, being the codec of choice for use in broadcast HDTV, HD movies (Blu-Ray) and internet streaming.

HEVC is essentially a continuation of H.264/AVC, similar in design, with added complexity in the processing to allow for up to a 50 % reduction in bitrate at same quality. One of HEVCs main goals is to allow the transition up to higher resolutions such as 4K.

Potential alternatives to HEVC are Google's VP9 [13] and the still-in-research codec Daala by xiph.org [14]. Both codecs aim to be just-as-good or better than HEVC while being royalty-free.

Still, 4K in distribution in the current state sounds unlikely. For distribution, we are still stuck with 720p or 1080i. Not even 1080p is standard, and a further jump to 4K/UHD sounds very unlikely to happen given that distribution links are very limited in bandwidth. If everyone moves to IP-based distribution to the home in the future, bandwidth can be increased over time, but we likely have to cater to limited-bandwidth transmission over the air in the near future.

Contribution

Broadcast contribution is the exchange of multimedia between professional users. A typical example is multimedia transport from events such as concerts and sporting events, covered by an OB-Van [1] or an internal production studio. Another example is transport of multimedia content between two studios. Multimedia content can be edited and post-processed in several stages at different locations before being broadcast to consumers. [15]

For contribution purposes, we have different requirements for video compression. Video quality must approach lossless. The first realization is that completely uncompressed video is not feasible when transmitting video over longer distances. Data rates for even the 720p60 10-bit 4:2:2 (1280x720) production format exceeds 1 Gbit/s, and it is obvious that uncompressed 4K video would be even more difficult to handle.

JPEG2000 [16] has emerged as a popular alternative for these use cases. JPEG2000 is a still-image codec based on the wavelet transform and was designed as a successor to the older JPEG standard. JPEG2000 and JPEG are international standards defined by Internation Organization for Standardization (ISO). In 2013, a technical recommendation was released by the Video Services Forum [17]. The goal of the technical recommendation is to create a framework for HDTV contribution (up to 1080p60) where vendors of contribution technologies can be interoperable with JPEG2000 technology. [18]

While JPEG2000 encoding can be done in 4K, it is challenging. JPEG2000 is a complex codec, with very advanced entropy coding which is difficult to implement at bitrates beyond 1 Gbit/s, especially with low latency ($<$ 1 frame).

Production

In production, raw and uncompressed video is still by far the most common way to transmit video information. The requirements on latency (ultra-low latency) and quality are such that compression has not been very practical. There have been attempts, and allowing HD production video to be transmitted over SD-SDI (270 Mbit/s) channels was a fairly popular way of reusing older SDI equipment during the transition to HD, one example being Dirac Pro [19]. With the transition to 4K and beyond, we will likely see a similar situation again.

[1] Outside broadcast van

In 2013, intoPIX announced a new codec - TICO [20] - which attempts to allow for compression in the production space. It targets 1:2 to 1:4 compression while being visually lossless. intoPIX states that TICO can scale to 4K and beyond.

1.2.2 Networking and transmission technology

For distribution of content to consumers, transmission is done at a fairly modest bitrate, $\approx$ 10 Mbit/s for TV channels. The challenge with broadcast distribution is that bandwidth is at a premium, and squeezing more content into fixed-bandwidth channels is challenging. 4K, with its increased resolution will certainly require more bitrate than current HD broadcasting. However, for the purposes of this section, we only concern ourselves with transmission technology in the production domain where we concern ourselves with physical channels. In order to transmit ever-increasing datarates over a physical link, networking and transmission technology must evolve along with it. In production, where only the highest quality is acceptable, the boundaries for transmission technology is pushed.

Serial Digital Interface

In HD production today, HD-SDI is ubiquitous. 1.485 Gbit/s HD-SDI can support 720p/1080i/1080p30 video formats, while a less common 2.97 Gbit/s interface can support full 1080p60 video signals with 10-bit 4:2:2 sampling. A large chunk of this bandwidth is reserved for blanking intervals, ancillary data and audio.

For 4K to be possible over SDI, newer standards are in the making, with support for 6 Gbit/s and 12 Gbit/s.

Ethernet

Ethernet is a very common interface, used for transmission of IP packets. Typical configurations support 1 Gbit/s, 10 Gbit/s, as well as possibilities for 40 Gbit/s and 100 Gbit/s (although far less common).

In the industry, there is a growing desire to move as much as possible over to IP-based solutions to avoid a plethora of dedicated communication equipment [2]. [21, 22] With 10 Gbit/s ethernet links, we could either multiplex several regular HD-SDI streams into the link or one 4K stream. Another use case is transmitting one HD-SDI stream over a 1 Gbit/s ethernet link, which is commodity even in the consumer market.

Both 4K over 10 Gbit/s and HD over 1 Gbit/s do not fit exactly inside a single link. To enable this, we need a light compression scheme which can reduce the bitrate by roughly 50 %. By compressing further, we could fit more streams inside a single link. Packet based ethernet solutions makes multiplexing and managing these streams easy compared to the very ”raw” serial format of HD-SDI. Multicast transmission is also possible with IP and Ethernet which enables some interesting applications.

1.2.3 Display technology

Display technology for 4K is available from many vendors, albeit still costly. It is being marketed actively as a new feature by TV manufacturers. It is likely that we will have to see widespread adoption of 4K diplays before broadcasting in 4K becomes viable.

[2]In the desktop PC market these days, we rarely see RS-232, parallel ports and special-purpose connectors for example. USB is everywhere.

We saw a similar situation with the transition from SD to HD. Long before broadcasting in HD was commonplace, the push to HD was motivated partly by gaming consoles as well and Blu-Ray video. For many, the transition from SD to HD also meant upgrading from older CRT TV sets to LCD or plasma, which were already HD-capable anyways.

Display manufacturers have begun to see the need for compression as the bandwidths required to transmit 4K and beyond over the consumer-oriented HDMI and DisplayPort at high frame rates is becoming more and more difficult. VESA recently released a specification for a new visually lossless, low-latency compression standard for display devices [23]. The recent HDMI 2.0 standard introduced support for 4K@50/60 with a total bandwidth of 18 Gbit/s. [24]

1.3 The light compression trend

It is clear from recent announcements of TICO and VESA Display Stream Compression that light compression will probably be a new trend in video compression. Transmitting uncompressed 4K data (10 Gbit/s and beyond) is taxing for hardware and the industry sees that adding some compression is easier than transmitting ever-increasing datarates.

1.4 Creating a codec from scratch

In order to meaningfully evaluate technology requirements for adapting HD-SDI to packet based IP networks, we either need to study an existing codec design (chapter 3) in detail and/or design a new codec based on experience from other codecs.

At the start of the project, three codecs were available to us as potential candidates for study, Dirac Pro [25] by the BBC, TICO [20] by intoPIX and VESA Display Stream Compression [23].

Between the three, TICO was found to be a codec targeted specifically towards the ultra-low latency use case in production. Unfortunately, a TICO implementation was not available for evaluation without signing non-disclosure agreements and for the purposes of this project would be meaningless as results could not be published.

Specifications for VESA Display Stream Compression are so recent that no implementation is available.

An open source implementation of Dirac Pro is available, with libschroedinger. Its low-latency implementation was found very buggy however, and for comparison or evaluation it could not be used. It was not possible to get in touch with libschroedinger developers.

Due to the apparent lack of easily available solutions to the given problem, we designed, specified and implemented a new video codec which aims to fulfill the challenges of ultra-low latency compression of HD-SDI signals at 4K and beyond. By developing a new codec from the ground up, we have the option to study the effect of each codec component more easily and draw more meaningful conclusions.

The Linelet codec developed here was inspired from publically available information on TICO as well as ideas from Dirac.

1.5 Roadmap for this project

In this project we have designed, specificed and implemented a codec from the ground-up and performed subjective evaluation with experts in the field. The project was conducted roughly in this order:

- Look at existing codecs in the lightweight compression space, identify their common features.

- Realize that neither of them had any readily available public solution which could be tested. Instead, design a codec based on similar principles, find possibilities for improvements.
- Implement this codec in software with the C programming language, create a public API for it (liblinelet) and implement support for Linelet compression in libavcodec/FFmpeg [3] for convenient testing.
- Test and refine the codec design on various test sequences, finding a sweet spot for psychovisual tuning.
- Add regression tests and optimize the software implementation for multithreading and SIMD for an overall $\approx$ 12-16x increase in performance over a simple C solution.
- Implement a command-line interface for convenient encoder and decoder support as well as tools for measuring PSNR, SSIM and similar.
- Design a subjective test of Linelet based on guidelines from ITU-R BT.500 [26] with help from experts in the field.
- Perform a subjective evaluation with experts using some critical test sequences.
- Evaluate the results from said test.

1.6 Thesis layout

In chapter 2, we present the basic theory for image and video compression. We look at the state-of-the-art of ultra-low latency video compression in chapter 3. The Linelet codec is described in chapter 4.

Testing methodologies for objective and subjective evaluation are found in chapter 6 and chapter 5 respectively. Results for both objective and subjective evaluations are presented in chapter 7. Discussion and conclusion are finally presented in chapter 8 and 9 respectively.

[3]http://www.ffmpeg.org/

Chapter 2

Video compression

In this chapter, we aim to explore the basics for various techniques which are commonly used for image and video compression today, creating a theoretical basis for the development of the Linelet codec (chapter 4). It is expected that the reader has basic knowledge of digital signal processing.

We also aim to look at various types of video codecs available today, study their strengths and see where Linelet fits in (section 2.8).

2.1 Introduction

Video compression deals with the compression of still images put in a sequence. Theory for image compression also applies to video compression.

Images and videos tend to contain very redundant information. The main theme of image and video compression therefore revolves around exploiting various kinds of redundancies.

Only exploiting the redundancies directly leads to *mathematically lossless compression*, where we can reconstruct an image or video sequence perfectly equivalent to the original, but (hopefully) using fewer bits than the original source.

In *lossy compression*, we reduce bitrate further by introducing errors in the reconstruction. If errors are introduced in a way that is not normally perceptible by humans, we call it *visually lossless compression*.

2.1.1 Spatial redundancy

In a still image, neighboring pixels are usually highly correlated. Exploiting this redundancy is the goal of signal decomposition (section 2.2).

2.1.2 Temporal redundancy

In video, there is very significant redundancy between successive frames which can be exploited. Video codecs which exploit temporary redundancy typically fall into the hybrid video codec class (section 2.8.1). It is not always desirable to exploit this redundancy due to its complexity.

2.1.3 Spectral redundancy

The most common way to represent color digitally is using the RGB (red, green, blue) system. However, there is some redundancy between these color channels which can be exploited.

2.1.4 The human visual system

The human visual system (HVS) is not perfect, and it has several factors which when taken into account can allow more compression.

- Less sensitive to color than intensity
- Less sensitive to high spatial frequencies

This in turn means we can treat more video information as redundant which allows better compression.

2.2 Signal decomposition

Signal decomposition attempts to exploit spatial redundancies in an image. In image and video codecs, we typically employ a time-to-frequency transform as we realize that highly correlated pixels translate to energy concentrated in the lower-frequency bands.

In essence, we utilize redundancy in the signal by splitting the information into an *important* part (low-pass) and a *less important* part (high-pass). This step is lossless, but it is vital to achieve good compression.

Popular alternatives for this kind of approach are based on Discrete Cosine Transform (DCT) or Discrete Wavelet Transform (DWT). Both transforms work in the frequency domain, but in significantly different ways.

2.3 Discrete cosine transform

The Discrete Cosine Transform [27] expresses a sequence of data as a sum of oscillating cosine functions at different frequencies (including DC). It is a special case of the Discrete Fourier Transform (DFT) for real data. Symmetric extension is assumed which makes the resulting coefficients real.

The DCT has a property of energy compaction, which means the energy of the resulting coefficients is heavily concentrated in a few coefficients around the DC component. The DCT is a very good approximation to the theoretical Karhunen-Loève transform (KLT) which optimally decorrelates coefficients. The DCT approximation only holds for highly correlated samples, which is the case for most still images.

Applying the DCT to an entire image is not computationally practical, so in order to apply the DCT to an image, one typically subdivides the image into N-by-N blocks. These blocks are then independently transformed. Typical block sizes range from 4x4 to 16x16.

The one dimensional forward DCT can be expressed as such:

$$X_k = \sum_{n=0}^{N-1} x_n \cos\left[\frac{\pi}{N}\left(n+\frac{1}{2}\right)k\right] \quad k = 0,\ldots,N-1 \tag{2.1}$$

For two-dimensional transforms, DCT transforms are separable just like the DFT, and 2D transforms can be achieved by transforming vertically and horizontally. After transforming an 8x8 block, we get 64 coefficients which represent basis functions which oscillate at different frequencies (figure 2.1).

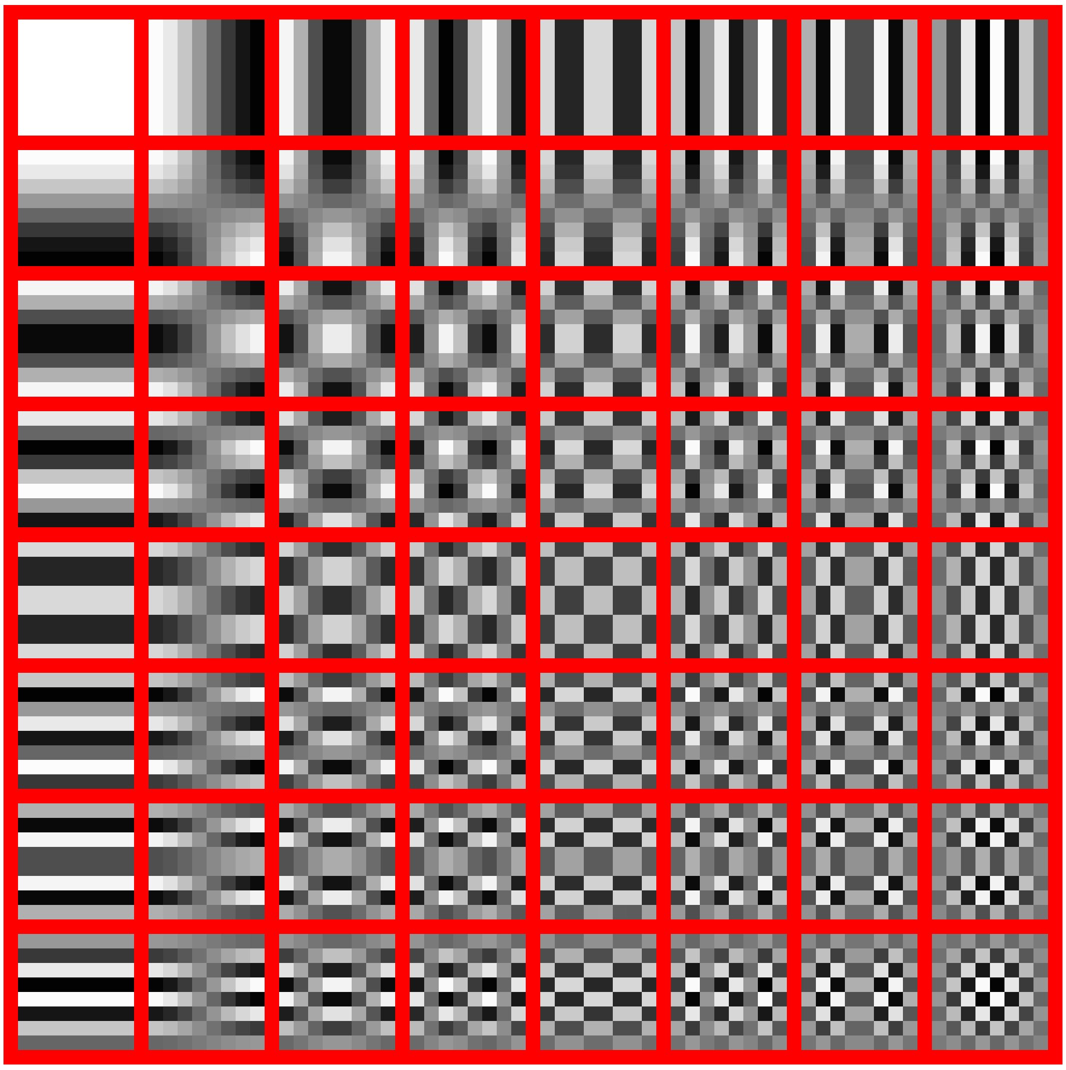

Figure 2.1: Basis functions for the 8x8 2D DCT [1]

Just like the fast fourier transform, a fast cosine transform optimization exists as DCT is just a special case of DFT.

The inverse DCT transform takes the coefficients and adds the 64 different basis functions together with weighting factors represented by the DCT coefficients.

2.3.1 Frequency vs. time resolution

Just like the discrete fourier transform, the choice of block length determines time and frequency resolution.

Modern DCT-based codecs allow the block size to vary, which can allow a codec to select whether or not time resolution (smaller block sizes) or frequency resolution (larger blocks) is most important. It is up to an encoder to find the best choice of block sizes.

2.3.2 Integer approximations

While the conventional DCT is defined with real numbers (cosine), integer approximations exist, which allow perfect reconstruction.

H.264 and HEVC use the *integer transform* which is an approximation to the DCT. Perfect reconstruction is important as it allows H.264 and HEVC video streams to have exactly *one* correct, compliant output after decoding. When using the floating point DCT one needs to make approximations, and some rounding errors between implementations must be assumed.

2.3.3 Blocking artifacts and de-blocking

A common problem with block-based DCT codecs (block based in general) is that transforming blocks independently can cause discontinuities when the blocks are quantized and transformed back to the spatial domain (figure 2.2).

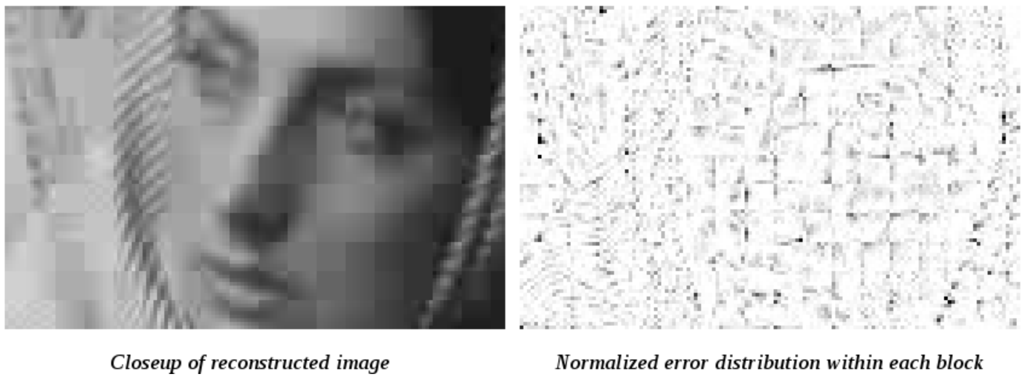
Closeup of reconstructed image
Normalized error distribution within each block

Figure 2.2: Illustration of severe blocking artifacts [2]

To remedy this, a deblocking filter can be run on the block boundaries after decoding. The deblocking filter must be adaptive and try to determine if the discontinuity observed was caused by quantization or if it was caused by actual details in the source image.

2.3.4 Lapped transform

A way of avoiding blocking artifacts is letting the DCT transform basis functions overlap, i.e. using basis functions which overlap into other blocks. The use of an overlapped DCT has been common in audio codecs by the modified discrete cosine transform (MDCT), but its use in video codecs is not very common. The JPEG XR ISO standard, as well as the upcoming Daala codec from xiph.org use some form of overlapped DCT transform [14].

2.4 Discrete wavelet transform

The discrete wavelet transform [28] can be seen as a 2-band sub-band coding technique. The wavelet transform low- and high-pass filters the input signal. After filtering, the low- and high-passed signals are then decimated by two. Using the two decimated bands, it is possible to recover the original signal by interpolating and applying synthesis filters as in figure 2.3.

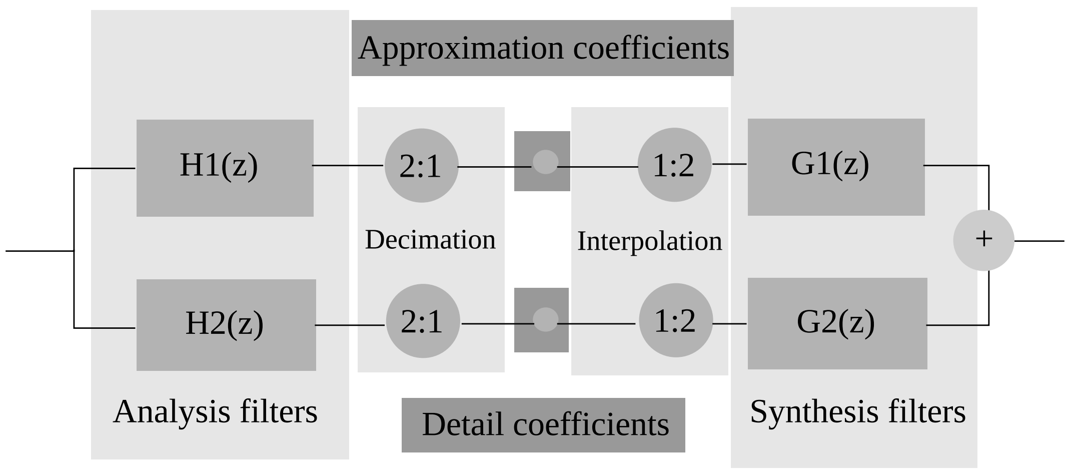

Figure 2.3: Critically sampled filterbank

The low- and high-pass filters used are designed specifically to allow recovery of the source signal even after decimation. Decimation adds aliasing to the two resulting signals, but the aliasing effects can cancel each other out after synthesis.

If there are strong correlations in the input signal, the power of the resulting high-pass band is significantly lowered.

For two-dimensional signals like images, the transform is applied in two directions with low- and high-pass filters being applied vertically and horizontally, which gives four different sub-bands instead of two.

For images, there are still significant redundancies left in an image after one transform. For further decomposition, only the low/low-pass band (LL band) is filtered as the high-passed coefficients are usually fairly well decorrelated already.

Figure 2.4 shows an image transformed with DWT in two dimensions. The resulting LL band is transformed again into four bands which gives us two decomposition levels.

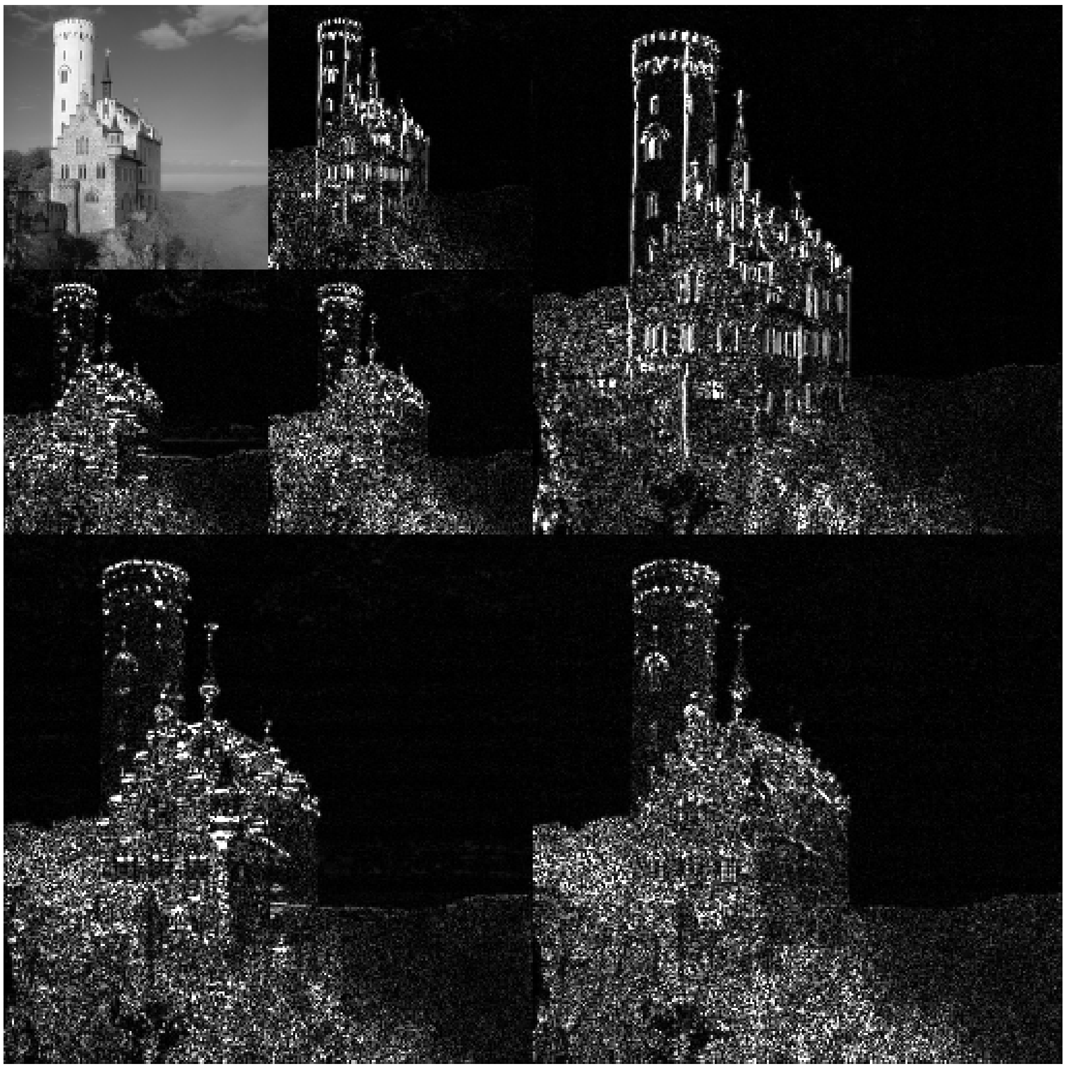

Figure 2.4: A 2-level 2-dimensional DWT [3]

2.4.1 Naming convention of wavelet subbands

With wavelet transforms, it is necessary to have a structured way of referring to the individual sub-bands by name. In this report, we will use the convention of referring to sub-bands by decomposition level and sub-band within that decomposition level.

With one level of decompositon, we have LL_1, LH_1, HL_1 and HH_1 bands. The first letter refers to the horizontal sub-band and second letter the vertical sub-band. I.e. after a horizontal low-pass and vertical high-pass we have the LH band. For each decomposition level, we increase the index by one. If we want two decomposition levels, we can transform the LL_1 sub-band, and obtain LL_2, LH_2, HL_2 and HH_2 bands. As these four bands allow us to recover LL_1, we can discard LL_1.

2.4.2 Frequency vs. time resolution

A property of the wavelet transform is that frequency and time resolution varies. In the highest frequency band, the frequency resolution is very low (half the original spectrum), but time resolution is very high. In progressively lower frequency bands, the time resolution lowers while frequency resolution increases (figure 2.5). This follows the uncertainty principle of signal processing that we cannot obtain good frequency resolution and time resolution simultaneously.

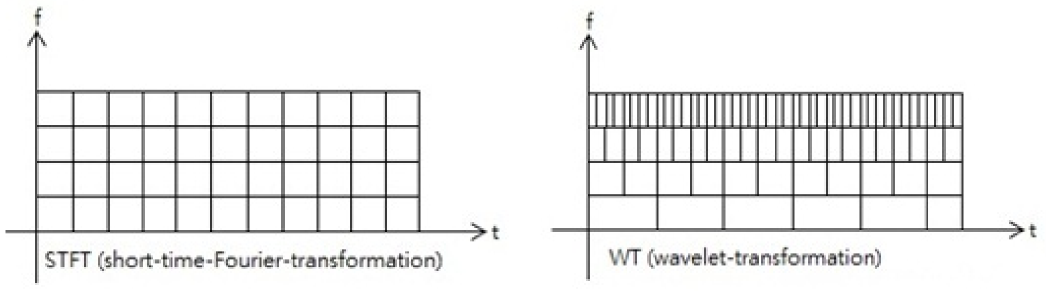
f
t
STFT (short-time-Fourier-transformation)
f
t
WT (wavelet-transformation)

Figure 2.5: Frequency vs. time resolution for fourier transform and wavelet transforms [4]

In practice, this means that wavelet transforms and DCT have different characteristics when faced with detailed parts (high frequency) of an image. DCT can express a high-frequency texture with a single coefficient due to its high frequency resolution but low time resolution.

With its high time resolution at higher frequencies, edges can be represented with a few wavelet coefficients. Sharp transitions tend to cause all coefficients in a DCT block to be affected as the frequency response of a delta-impulse is uniform for all frequencies. This effect can often be seen as *mosquito noise* in DCT-based codecs (figure 2.6). With DWT, the resulting artifacts (figure 2.7) are not ”randomly” spread around like in JPEG, but are more compactly centered around the source.

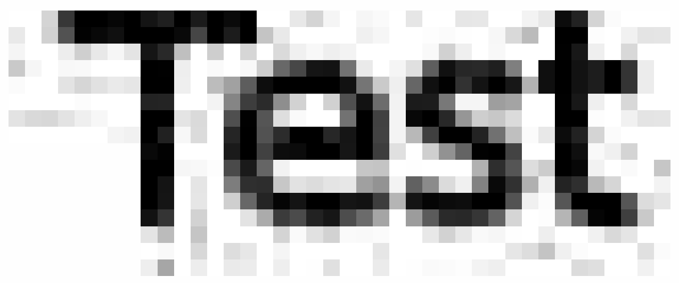

Figure 2.6: Effect of low time resolution in JPEG (8x8 DCT)

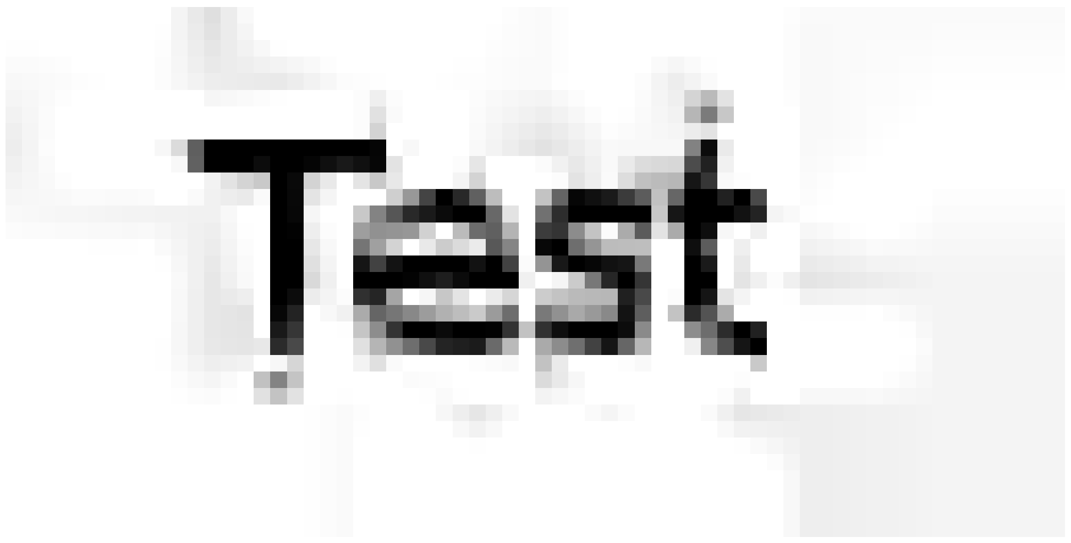

Figure 2.7: Effect of higher time resolution in JPEG 2000 (DWT)

2.4.3 Advantages of DWT

Flexibility

Wavelet transforms are flexible in that the choice of filters is arbitrary (section 2.4.5).

Scalability

Wavelet transforms are *scalable* in that during decoding of an image, one can stop decoding at the desired resolution, discarding higher resolution detail. E.g. we could simply discard the detail coefficients (figure 2.3) and only use the approximation coefficients.

Avoiding blocking artifacts

As the wavelet transform has no block boundary, blocking artifacts can be avoided entirely.

2.4.4 Disadvantages of DWT

Blurring artifacts

A common problem with wavelet-based compression is blurring. The blurring is introduced if the power of high-frequency information is reduced too much during quantization (section 2.7). Zeroing out the high-pass band is equivalent to low-pass filtering the original image, which shows up as blurring.

Ringing artifacts

Depending on the wavelet filters used, ringing around edges can be a common artifact. The wavelet filters used are usually ”ringy” in nature (negative values in filter coefficients), and a high-pass wavelet coefficient represents a small ”ringing” region of the image. Introducing error to such a wavelet coefficient is thus the same as adding a small ringing function to the resulting image.

2.4.5 Common wavelet filters

Haar wavelet

The Haar wavelet is the simplest wavelet transform possible. It consists of taking the average (low-pass) and difference (high-pass) of two samples.

$$H_1(z) = \frac{1+z}{\sqrt{2}} \tag{2.2}$$

$$H_2(z) = \frac{z-1}{\sqrt{2}} \tag{2.3}$$

While wavelet transforms in general avoid blocking artifacts, the Haar wavelet does not avoid it. A decimated approximation sample and detail sample is only influenced by two samples, meaning that two neighboring filtered samples are not computed from the same samples, i.e. a block transform.

Le Gall 5/3 wavelet

The 5/3 Le Gall wavelet is a reversible wavelet transform. It is particularly popular for mathematically lossless compression. It is also popular for its very simple implementation, which requires no multiplications to implement. It is the default filter used for lossless compression in JPEG2000 [16].

$\lvert n\rvert$	Analysis H_1	Analysis H_2	Synthesis G_1	Synthesis G_2
0	6/8	1	1	6/8
1	2/8	-1/2	1/2	-2/8
2	-1/8			-1/8

Table 2.1: Le Gall 5/3 wavelet filter coefficients

CDF 9/7 wavelet

The CDF 9/7 wavelet is an irreversible wavelet transform due to its floating point definition. It is however recognized as a superior filter for lossy compression. It is the default filter used for lossy compression in JPEG2000.

$\lvert n\rvert$	Analysis H_1	Analysis H_2	Synthesis G_1	Synthesis G_2
0	+0.60294	+1.11508	+1.11508	+0.60294
1	+0.26686	-0.59127	+0.59127	-0.26686
2	-0.07822	-0.05754	-0.05754	-0.07822
3	-0.01686	+0.09127	-0.09127	+0.01686
4	+0.02674			+0.02674

Table 2.2: CDF 9/7 wavelet filter coefficients

2.4.6 Signal extension

When applying the wavelet transform at image boundaries, one must decide how samples outside the image boundary should be treated. The method employed by JPEG2000 involves mirroring input samples, i.e.

$$s_{ext}(n) = \begin{cases} s(-n) & \text{if } n < 0 \\ s(2n_{max} - n) & \text{if } n > n_{max} \\ s(n) & \text{otherwise} \end{cases} \tag{2.4}$$

where $s(0)$ is the first sample.

2.4.7 Lifting implementation of wavelet transforms

Lifting is a method to more efficiently implement the filter bank used in wavelet transforms. The lifting method updates input samples with filtered versions of other samples. These filtered samples are then reused to compute and update other samples, where each iteration is called a *lift*. This effectively reduces required computation by a factor of 2 compared to a convolution and decimation approach as illustrated in figure 2.3.

Lifting also has the advantage of allowing an in-place transform, without consuming extra memory. Lifting methods also ensure that the transform is perfectly reversible as one can simply do the lifting step in reverse order to get back to the original input.

The general lifting approach

Assume a block of N samples after lifting step l, $x^l[n]$. For each lifting operation, we update one of the variables based on other samples and other samples remain unchanged. The functions $f^l(\cdot)$ can be arbitrarely chosen. When updating $x^l[n]$, it is also possible to invert the result, as noted with the k^l factor.

$$x^{l+1}[n] = k^l \cdot x^l[n] + f^l\left(\{x^l[m]\}, m \in [0, N-1], m \neq n\right),\ k \in \{-1, 1\} \tag{2.5}$$

$$x^{l+1}[m] = x^l[m], m \neq n \tag{2.6}$$

To reverse the operation, we simply step backwards, reversing all lifting steps.

$$x^l[n] = k^l \cdot x^{l+1}[n] - k^l \cdot f^l\left(\{x^{l+1}[m]\}, m \in [0, N-1], m \neq n\right),\ k \in \{-1, 1\} \tag{2.7}$$

$$x^l[m] = x^{l+1}[m], m \neq n \tag{2.8}$$

While the wavelet filter coefficients in section 2.4.5 are defined in terms of convolution, it is possible to find a lifting style implementation for them.

2.5 Prediction filter

In most block-based DCT codecs, a prediction filter is used in cooperation with DCT to decorrelate over block boundaries. The prediction is usually done in the spatial domain, before DCT of prediction error is performed. Several mathematically lossless video codecs use prediction filters exclusively to exploit redundancies.

Prediction filters generally work by knowing previously decoded symbols (causal neighbors). E.g., a simple prediction filter can work by subtracting the mean of neighboring causal symbols

$$\hat{P}(x, y) = \frac{P(x-1, y-1) + P(x, y-1) + P(x-1, y) + P(x+1, y-1)}{4} \tag{2.9}$$

One would then encode the prediction error $P(x, y) - \hat{P}(x, y)$ instead of just $P(x, y)$. The decoder can reverse this by adding the prediction error to the predicted value.

To ensure correct decoding, the prediction must happen in-loop, i.e. the values used for prediction must be the actual (distorted) decoded samples which is the only data the decoder has access to.

A downside with prediction is that it enforces a certain order on the encoding and decoding, which can make parallelization techniques less efficient.

Prediction doesn't necessarily have to be done between pixels of the same frame (intra-prediction), it can also be done between frames of video (inter-prediction, motion compensation). Prediction between frames is such a good way to exploit redundancy that hybrid video codecs [1] are in a class of their own (section 2.8.1).

2.6 Entropy coding

After transform, prediction and quantization (section 2.7), the resulting coefficients (symbols) should be encoded using as few bits as possible. The information entropy formula by Shannon [29] describes the theoretical boundary for lossless compression when the probability distribution of symbols is known.

$$H(X) = E\{I(X)\} = \sum_{i \in A_x} P(i) \cdot \log_2 \left(\frac{1}{P(i)} \right) \tag{2.10}$$

where $H(X)$ is the minimum number of bits needed per symbol on average and $I(x)$ is the information gained by seeing symbol x. Rare symbols contain more information, and common symbols contain little information. Packing coefficients in the bitstream while approaching the theoretical optimum is the goal of the *entropy coder*.

2.6.1 Variable length coding

A well known example of variable length coding is the Huffman code. The Huffman code assigns a fixed number of bits to every symbol. Common symbols are coded with fewer bits and uncommon symbols are coded with more bits. Table 2.3 shows the optimal encoding of a simple distribution.

Symbol	Probability	Code (binary)
0	1/2	0
1	1/4	10
2	1/8	110
3	1/8	111

Table 2.3: Optimal variable length code for a simple distribution

The weakness of variable length coding is that one must spend at least 1 bit to encode a symbol. This can be a great source of redundancy when the entropy $H(X)$ is very low. A way to work around this limitation is to encode more symbols at a time to ensure that no single ”super-symbol” have entropy less than one bit, e.g. creating a codebook which considers all permutations of N concatenated symbols. The limiting factor however is that the code alphabet grows exponentially.

Another weakness is that variable length codes cannot easily adapt to changing probability distributions. The code is typically precomputed with an assumed probability distribution. To adapt, one could compute several different code books and dynamically change these while coding, but this also does not really scale beyond a couple codebooks.

2.6.2 Arithmetic coding

Arithmetic coding is a more efficient and computationally expensive compression algorithm.

The main improvements of arithmetic coding over variable length coding are that symbols can be coded with fractional bits. Additionally, the estimation of symbol probabilities can be done dynamically without recomputing any tables.

[1]Hybrid refers to both transform coding and prediction based coding

The main idea of arithmetic coding is to encode incoming symbols as two numbers between 0 and 1 with infinite precision. The difference between the two numbers express the probability of having seen all the accumulated symbols so far. The arithmetic encoder encodes a binary number range which unambiguously lies between the two numbers. The maximum number of bits required to encode this range is equal to

$$B = \left\lceil \log_2 \left(\frac{1}{P(x)} \right) \right\rceil + N \tag{2.11}$$

which shows that as the message grows large, the constant N term becomes insignificant, and we can approach the theoretical optimal.

To make the algorithm practical, several approximations must be made. Finite precision of probabilites along with finite precision of the accumulated probability ensures that some redundancy must be accepted.

If the symbols to be encoded are binary, a look-up table simplification can be made. All current video compression standards encode binary decisions. A well known example is the CABAC encoder found in H.264/AVC and H.265/HEVC.

2.6.3 Estimating probabilities

Optimal entropy coding relies on knowing the correct probability distributions. In practical video compression, one can only get a rough estimate of these distributions, and distributions are not constant across a video or image.

Arithmetic coders usually solve this by using a frequency count estimate or a finite state machine. Encoders and decoders must have the exact same probability estimates for this to work, so the exact probability estimation method must be rigorously defined by a codec.

2.7 Quantization

After transforming and predicting a video frame, the redundancy has been reduced, but it is inherently a lossless process. An inverse transform would perfectly recover the original data assuming the transforms are exact reversible.

To reduce the entropy of the resulting signal even further, the coefficients are quantized. This allows the coefficients to be represented with fewer bits. Both the DCT and DWT transforms tend to make most high-frequency samples close to 0. Quantizing these coefficients often produces large strings of zeroes, which is very beneficial for entropy coding later on.

Quantizers can be linear or non-linear. For video compression, linear quantizers tend to be most useful as they are very simple to implement. Two well known non-linear (logarithmic) quantizers include a-law and μ-law, schemes used for telephony.

2.7.1 Round-to-nearest

A common linear quantizer. If Δ is power-of-2, it can be implemented as a binary shift operation.

$$Q(x) = \left\lfloor \frac{x}{\Delta} + \frac{1}{2} \right\rfloor \tag{2.12}$$

$$\hat{x} = Q(x) \cdot \Delta \tag{2.13}$$

2.7.2 Deadzone

As entropy coding favors long strings of zeroes, it can be beneficial to force values which come close to 0 after quantization to 0 for the purposes of avoiding breaking a long string of zeroes.

$$Q(x) = \text{sign}(x) \cdot \left\lfloor \frac{|x|}{\Delta} \right\rfloor \tag{2.14}$$

This quantization approach ensures that the region around 0 is twice as large as for any other quantization level. In the literature, a deadzone quantizer is good for lower bitrates, while uniform quantization is very-near optimal for higher bitrates. [30]

Deadzone adds more distortion than a round-to-nearest quantizer due to the larger region around 0, but ideally, it will reduce entropy better than round-to-nearest such that the rate-distortion performance is overall improved.

Dequantization of deadzone is slightly more complicated than round-to-nearest as the result depends on the sign.

$$\hat{x} = \text{sign}\left(Q(x)\right)\left(\left(|Q(x)| + \delta\right) \cdot \Delta\right) \tag{2.15}$$

where δ is generally 0.5. It is important to clarify that $\text{sign}(0) = 0$.

2.7.3 Vector quantization

Vector quantization is a method where multiple input values (vector) are mapped to a single quantized index.

In theory, vector quantization can exploit redundancies between values and thus obtain better rate-distortion properties than scalar quantization. It is possible to obtain theoretically optimal rate-distortion by use of vector quantization only without signal decomposition, but this is mostly a theoretical result as one will potentially need very large vectors, and even larger code books which scale exponentially in size with vector length.

In practice, the initial signal decomposition step is assumed to mostly decorrelate all samples. When values are perfectly decorrelated, simple scalar quantization is just as good as vector quantization and far simpler.

2.8 Video compression tradeoffs

In video compression, there are several trade-offs which must be made in the design, which makes creating the ultimate video codec for all cases almost impossible.

Some desirable properties of video codecs are:

- Low complexity
- Good compression vs. quality
- Low latency
- Deterministic and simple rate control

As there are different use cases for video compression, several types of video codecs exist which attempt to optimize for their particular use cases.

2.8.1 Hybrid video codecs

Hybrid video codecs attempt to optimize the compression rate down to bitrates suitable for media end-consumers (1:100 to 1:1000 compression rates). To achieve this degree of compression, correlation in-between frames (temporal redundancy) is exploited using motion compensation, a method where a predicted frame is synthesized from other frames with offsets which correspond to motion in the video.

The name *hybrid* comes from the combination of a transform codec and a predictive codec. The current frame is predicted from previous (and/or future) frames with motion compensation, and the prediction error is encoded as a still image with transform coding. To allow seeking and error recovery, frames have to be encoded without reference to other frames (intra) at regular intervals.

Hybrid video codecs are almost always based on the DCT, as the block based structure fits well with motion compensation, which is also performed in a block-based fashion. Dirac is a notable exception to this, which uses DWT with overlapped block motion compensation (OBMC), but it has been found not competitive with recent DCT-based hybrid video codecs.

There have been attempts at 3D wavelets - wavelets which also filter temporally - but it has not been very successful.

Hybrid video codecs have very good compression, but tend to have high complexity, high latency and non-trivial rate control. The drawbacks can be reduced by sacrificing compression performance.

Some hybrid video codecs

- H.265/HEVC
- H.264/AVC
- Theora
- Daala (research)
- VP8/VP9
- Dirac

2.8.2 Mathematically lossless intra

This type of video codec is designed to be an alternative to storing raw uncompressed video. These are typically used for archival and/or video editing purposes and are useful in cases where only 100% perfection in reproduction is allowed.

The focus of these codecs is supporting high bit-depths, and to have good compression/speed tradeoffs. They generally do not exploit temporal redundancies as it makes working with the image material difficult. If one were to change one frame, many encoded frames would be affected as they all depend on each other via prediction.

Some lossless intra codecs

- FFV1
- HuffYUV
- H.264/AVC, using QP = 0
- UTVideo
- MagicYUV

2.8.3 Broadcast contribution

Broadcast contribution is a class of video codecs that are designed to transmit very-high quality video between professional producers and consumers of video. Cases here can include live feeds from sporting events and concerts as well as transmission of material in-between studios. Compression rates here are typically 1:10 (100-200 Mbit/s for HD) [31].

Codecs for contribution must allow repeated encoding/decoding without significant loss in quality [32], have low latency (approx. 1 frame or less), reasonable complexity and deterministic rate control. Degradations in quality should also not significantly affect the result of the final encode targeted end-consumers.

Due to the requirement of repeated encoding/decoding, contribution codecs are intra-only. Hybrid video coding is not very practical for this use case as editing a single frame requires decoding and re-encoding an entire GOP.

Some broadcast contribution codecs

- JPEG2000 broadcast profile
- H.264 intra/AVC-I

2.8.4 Transmission media adaptation

Transmission media adaptation is a class of video codecs where very low latency ($\leq$ 1ms), very low complexity and deterministic rate control are top priority. This completely sacrifices compression performance, but these codecs compensate by operating at very high bit rates (typically 1:2 to 1:4 compression ratios).

Proposed use cases for this class of codecs is adapting HD-SDI signals to packed based IP networks, or lower bit-rate requirements for transmission equipment at higher resolutions.

Some transmission media adaptation codecs

- Linelet
- TICO
- Dirac Pro - Low Delay
- VESA Display Stream Compression

2.9 Rate control and latency

In real-time scenarios such as contribution, transmission media adaptation and real-time streaming, a maximum latency constraint as well as a limited bandwidth constraint must be met at the same time.

A problem with entropy coding is that one does not know the final bitrate until one has actually performed entropy coding. This has the effect that achieving the desired bitrate becomes a feedback system where one must monitor the resulting bitrates, and adapt accordingly. The adaptation might have to pessimize its estimation to ensure that maximum buffer size constraints are guaranteed to be met. In worst case, one might have to re-encode parts of the video to ensure that buffer size constraints are met, or simply discard data.

2.9.1 Constant bit rate

Constant bit rate allocates a fixed bandwidth per unit time. This ensures minimum latency, but quality cannot be constant. Constant bit-rate rate allocation aims to optimize quality within the bitrate constraint.

Embedded entropy coding

In many wavelet entropy coding schemes such as EZW [33], SPIHT [34] and EBCOT (JPEG2000) [16], a bitstream is organized such that a prefix of the stream is a lower-quality representation of the original bitstream. When such an entropy coding scheme is used, constant bit rate becomes trivial to implement as one can simply stop entropy coding when the desired bitrate is met. This scheme works by successively refining the resulting image where most significant information is moved first in the stream.

2.9.2 Variable bit rate with constrained buffers

Variable bit rate allows bitrates to temporarily increase beyond the average bitrate to be able to encode video with a more uniform quality. As the bitrate is allowed to increase, the receiver must account for increases in transmission time which adds latency as the receiver must maintain a sliding window buffer to soak up variability in the bitrate.

Allowing variable bit rates lessens the problem of rate control as bitrates are allowed to fluctuate slightly per frame. With hybrid video codecs, a scheme like this is natural as intra-only frames (I-frames) take up far more bandwidth than motion-compensated frames (P-frames).

2.9.3 End-to-end latency

When discussing end-to-end latency, there are several factors which come into play. Total end-to-end latency is the sum of all latencies in the entire chain (figure 2.8).

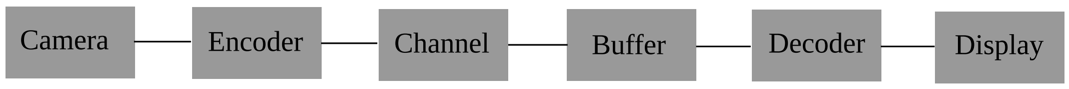

Figure 2.8: A real-time streaming system

- **Camera latency** - The latency from first scanline is seen at camera lens until the camera sends data to the encoder.
- **Encoder latency** - The latency from data is received at the encoder until it transmits encoded data for the input data.
- **Channel latency** - Network latency. Time from a packet is sent until it is received at the decoder.
- **Buffer latency** - A buffer is an intentional latency added to compensate for variable delay other places in the system. A display needs to output image data at a constant rate, but the streaming system might have jitter several places in the system making smooth video otherwise impossible.
- **Decoder latency** - Time for encoded data to appear at decoder until it is decoded and sent to buffer.
- **Display latency** - Latency of the display system.

For an encoder, camera and display latencies are irrelevant as they are fixed parameters outside the scope of a codec. The channel is also mostly irrelevant. The only restriction of the channel is the bitrate. If our average bitrate is less or equal to channel bitrate, we assume it adds a fixed latency.

This leaves encoder, decoder and buffer latencies. Depending on the rate control method used, we might require the use of a buffer. For variable bit rate with constrained buffers we intentionally add latency to be able to temporarily use higher bitrates than the average. To achieve minimum latency, we must use constant bit rate.

This leaves encoder and decoder latencies. These latencies are highly codec dependent, but we can study the latency of a hypothetical low-latency codec. We assume that the codec implementation is pipelined, i.e. the encoder and decoder do several steps in parallel at the cost of added latency.

The encoder pipeline

The encoder has three tasks:

- Receive input data
- Encode input data
- Transmit encoded data

If we assume the codec encodes one scanline at a time, a pipelined encoder architecture can do these steps simultaneously:

- Receive scanline N
- Encode scanline $N - 1$
- Transmit scanline $N - 2$

We see that the latency of this encoder is 2 scanlines. Instead of transmitting the ideal scanline N, $N - 2$ is transmitted. The time to transmit a scanline cannot exceed the time to receive one scanline or the pipeline would stall, so we assume constant bit rate.

The decoder pipeline

The decoder has three tasks as well:

- Receive data from buffer
- Decode input data
- Transmit decoded data to display

Again, we assume a pipeline, and 2 scanline latency. The overall latency is then 4 scanlines. For a (3840x2160) 4K video at 60 Hz we have a latency of 31 μs contributed from encoder and decoder alone for constant bit rate.

Chapter 3

State of the art, ultra-low latency codecs

During the initial research for codecs which would fit the problem description, three potential codecs were identified. Of these three codecs, only Dirac Pro has a publicly available specification which can be studied in detail. [35]

3.1 TICO

In 2013, intoPIX announced a new codec, TICO [20], which is designed to tackle the challenges of ultra-low latency compression at very high resolutions.

From publicly available information [36], the codec is based on compressing single scanlines. A Le Gall 5/3 wavelet transform is applied to the scanline horizontally. The resulting wavelet coefficients are represented with sign-magnitude. To meet target bitrates, least-significant bits can be discarded, which should be equivalent to deadzone quantization.

Wavelet coefficients are grouped in precincts. It is not clear from the presentation exactly how the wavelet coefficients are packed, but it appears there is no conventional entropy coding applied to the coefficients.

At the time, there is no publicly available data on how well the codec performs in practice. Nor is there publicly available software which can be used to make evaluations.

TICO is designed to be implemented on an FPGA. It does not require external memory and can be implemented very cheaply on minimal silicon.

3.2 Dirac Pro (VC2) - Low Delay

Dirac and Dirac Pro are codecs developed by the BBC [25]. The specification defines a hybrid video codec based on the discrete wavelet transform as well as a intra-only production oriented codec which is a subset of the full Dirac specification.

Dirac supports a wide range of wavelet filters from the simplest Haar transform to even longer wavelet filters than the CDF 9/7.

Dirac Pro supports a low-delay syntax profile which enforces a fixed bitrate per image slice, suitable for very low-delay operation. In addition, arithmetic coding is turned off. The slice structure appears to be fairly flexible, with possibilities of subdividing both horizontally and vertically. For very low-latency operation, one could use a slice height equal to a few scanlines.

Even in low-latency mode where arithmetic coding is turned off, Dirac Pro employs an exp-golomb variable-length code to encode wavelet coefficients.

Unlike TICO, Dirac Pro Low delay allows for vertical wavelet transforms as well as horizontal, which should in theory allow for better compression as vertical redundancies can be exploited as well.

Dirac also adds intra prediction of DC wavelet coefficients.

3.3 VESA Display Stream Compression

In 2014, VESA announced a new visually lossless compression standard designed for use in the physical link layer of transmitting video data to displays [23]. It is stated to operate at 8 bit/pixel, a compression ratio of 1:3 for 8 bit per channel video data (24 bpp).

The codec is based on DPCM, indexed color history (ICH), rate control and some form of entropy coding. It is based on adaptive prediction to handle challenges with computer graphics. Computer graphics tend to have many flat regions (very easy to code), but some regions with high amounts of detail which is difficult to encode with a single toolkit. Like TICO, it is scanline oriented. For colorspace transform, it uses the reversible YCgCo transform.

3.4 Comparison

These three codecs do not have any obvious similarity in the design. Two of them are based on wavelets (TICO, Dirac), and two of them are based on compressing single scanlines (TICO, VESA). Two of them have some kind of entropy coding scheme of coefficients (VESA, Dirac), but there is no codec design that applies to all of them.

In terms of simplicity, TICO is likely a winner. There is even less information on the internal design of VESA DSC, so it's not clear if VESA DSC is actually simpler.

Dirac, even in low-latency mode appears to be on the complex side. The use of an exp-golomb variable length code suggests that the Low Delay syntax is just a simplified version of the full Dirac specification.

Chapter 4

Linelet codec

4.1 Overview

The linelet video codec developed in this project is designed to address several goals which must be met to be useful for the purpose of adapting HD-SDI signals to ethernet and production.

- **Ultra-low latency**, end-to-end codec latency should be a few scanlines.
- **Ultra-low complexity**, the implementation should be simple enough to allow tiny hardware implementations and a throughput of several Gbit/s on off-the-shelf PC hardware.
- **Performance scalability**, the codec should facilitate a highly parallelized implementation on PC hardware (at cost of latency). Efficient GPGPU implementations should also be feasible. The design should also allow fine-grained parallelism to facilitate use of SIMD-like processing [1].
- **Deterministic rate-control**, the implementation should be able to easily target any bit-rate, e.g. an encoded scanline must be able to fit inside a fixed buffer size. Ideally, rate control decisions should be possible to make *before* any entropy coding is done.
- **Visually lossless compression**, only light compression rates are required to fit HD-SDI payloads into ethernet links, but the resulting artefacts must remain imperceptible to human eyes. Bitrates from 1:2 compression down to 1:4 compression should remain visually lossless for all sensible material.
- **Mathematically lossless**, the codec should be able to target mathematically lossless compression. When it is not possible to fit mathematically lossless data into buffers, the quality (visually lossless) should be gracefully degraded to meet required bit rates.
- **Robustness**, decoding and recompressing the video stream up to 7 times (EBU requirement) should be possible without any significant loss in quality. The codec processing should be designed so that errors do not accumulate significantly.
- **High bit-depth**, in production, 10-bit per color and beyond is commonly used. The codec should support higher bitrates with one common processing chain.

[1] Single instruction, multiple data

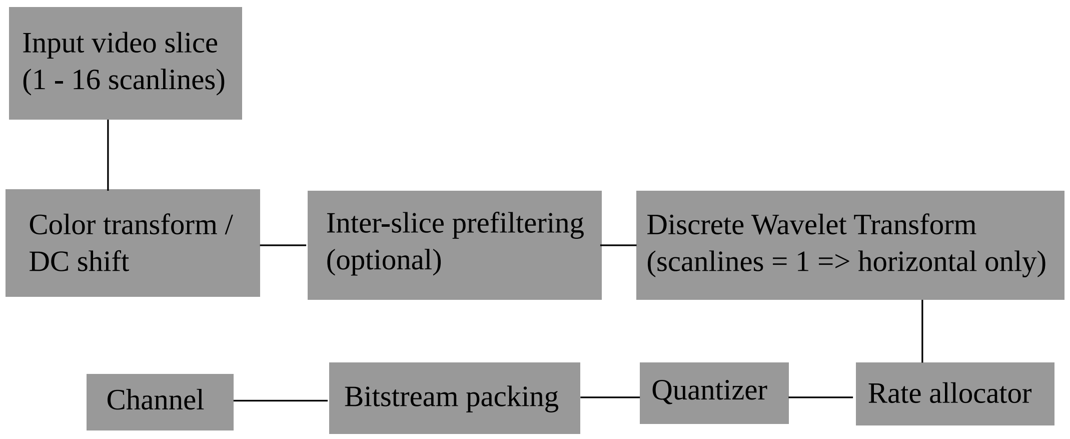
Input video slice
(1 - 16 scanlines)
Color transform / DC shift
Inter-slice prefiltering (optional)
Discrete Wavelet Transform (scanlines = 1 => horizontal only)
Rate allocator
Quantizer
Bitstream packing
Channel

Figure 4.1: Overview of the Linelet encoder

For Linelet, we chose a discrete wavelet based encoder. Each step of the encoder was designed with processing requirements in mind. Especially the bitstream packing was designed for optimal speed. It sacrifices compression efficiency for performance. The design of the bitstream packing also affects the rate allocator, which can target a fixed bitrate before any encoding is performed. Performing rate control before any encoding greatly reduces the complexity of the implementation.

All processing (except for quantization) in Linelet are all fully reversible ensuring fully lossless and lossy encoding with the same toolset. The fully reversible nature of Linelet also allows successive decoding and encoding to be performed with minimum additional loss (usually zero additional loss).

4.2 Splitting images into slices

To achieve ultra-low delay operation, the encoder must start sending out packets as soon as source image data is available. Due to bandwidth constraints of transmission cables and hardware, scanlines of the input image are assumed to be received in raster order one after another with a time delay.

If the encoder needs to wait until the entire image is received to begin encoding, an entire frame of latency is already added to the end-to-end latency.

To alleviate this, it was deemed necessary to split the source image into slices, sub-images which can be decoded and encoded independently. Slices are encoded and treated as separate entities which makes it feasible to begin encoding and transmitting as soon as the corresponding sub-images are received at the encoder.

The slice concept is found in several video codecs, e.g. in H.264, FFV1 and JPEG2000 (where the concept is dubbed *tiles*). While H.264 and JPEG2000s slices/tiles are very flexible in which parts of the image to put into a slice, a Linelet slice consists of a fixed number of scanlines.

Slices also allow completely parallel encoding of the input image. Fully parallel encoding however requires more slices to be buffered up at the same time, so the latency must increase. Parallelizing an encode on the slice level is however mostly useful for software implementations where ultra-low latencies are not possible or meaningful anyways.

The number of scanlines supported was chosen to be in the range of 1 to 16. Increasing the number of scanlines to 32 and beyond was not found to meaningfully increase compression performance. Increasing the possible number of scanlines makes hardware decoding costlier to implement (more memory required to decode all valid bitstreams) and latency would increase.

4.3 Colorspace

Linelet supports two representations of color images. While RGB is generally used as the final color representation on display devices, it is possible to exploit some redundancy between color channels by using a luminance/chroma representation (YCbCr).

There are several standardized ways for converting between RGB and YCbCr, which makes talking about YCbCr color spaces less useful without also specifing which particular *color matrix* is used.

4.3.1 Reversible color transform

The reversible color transform (JPEG2000 RCT) is designed to be able to losslessly represent RGB while getting some gain by reducing the redundancy between channels.

$$Y = \left\lfloor \frac{R + 2G + B}{4} \right\rfloor - 2^{\text{bits}-1} \tag{4.1}$$

$$C_b = B - G \tag{4.2}$$

$$C_r = R - G \tag{4.3}$$

The subtraction of $2^{\text{bits}-1}$ ensures that all components have a dynamic range centered around 0, which is useful as Linelet (and most wavelet codecs) operate on signed coefficients.

It is important to note that while the dynamic range of Y does not increase, the dynamic range of C_b and C_r must increase with one bit due to the subtraction.

The inverse can be computed as

$$G = Y - \left\lfloor \frac{C_b + C_r}{4} \right\rfloor + 2^{\text{bits}-1} \tag{4.4}$$

$$R = C_r + G \tag{4.5}$$

$$B = C_b + G \tag{4.6}$$

4.3.2 YCbCr

YCbCr is a representation commonly used in broadcast television (ITU-R BT.601 [37] and ITU-R BT.709 [6]). It is related to RGB with floating point coefficients, which makes the conversion between the two irreversible.

YCbCr as defined in BT.601 and BT.709 does not use the entire digital range of values. Some headroom at top (240-255) and bottom (0-15) is reserved for 8-bit video. Occasionally, the full range (0-255) is used. This also makes it necessary to specify *full-range* or *TV-range* (16-239) in some cases.

As Linelet wants input samples to have dynamic ranges centered around 0, input YCbCr values are simply DC-shifted as a preprocessing step.

$$Y' = Y - 2^{\text{bits}-1} \tag{4.7}$$

$$Cb' = Cb - 2^{\text{bits}-1} \tag{4.8}$$

$$Cr' = Cr - 2^{\text{bits}-1} \tag{4.9}$$

4.3.3 Subsampling chroma

When using a YCbCr representation, it is common to reduce the resolution of chrominance channels. This decision is made based on the human visual system, which is better at spotting details in luminance (gray tone) than color details. Three common formats for YCbCr are:

- **4:4:4** - No subsampling
- **4:2:2** - Horizontal chroma resolution halved
- **4:2:0** - Both horizontal and vertical chroma resolution halved

For production, 4:4:4 and 4:2:2 are the most common, while 4:2:0 is common for broadcast distribution. As Linelet is a scanline oriented codec, it was found worthwhile to support horizontal subsampling only, i.e. 4:4:4 and 4:2:2.

4.4 Wavelet transforms

Linelet uses a horizontal wavelet transform as well as an optional vertical transform. The Le Gall 5/3 filter is used both horizontally and vertically.

The very simple Haar wavelet was used for vertical transforms instead of Le Gall in the experimentation phase, but the 5/3 filter was found to achieve significantly better results at target bitrates. The Haar vertical wavelet was kept as an alternative to Le Gall in the implementation.

The vertical transform is considered of less importance as the number of scanlines is fixed from 1 to 16, which limits the number of vertical decomposition levels available. If there is only a single scanline in a slice, the vertical transform is disabled for obvious reasons anyways.

4.4.1 Lifting implementation of vertical wavelet

Vertical transforms in Linelet are implemented as an in-place lifting transform between scanlines.

The Haar vertical lifting step considers two vertically adjacent samples A and B which result in a low-passed sample A_l, and high-passed sample B_l.

After one level of decomposition has been computed, we only consider scanlines which contain low-passed A_l samples, and lift those further in a butterfly pattern (figure 4.2). For the 5/3 filter, we employ the scheme in figure 4.3, just vertically.

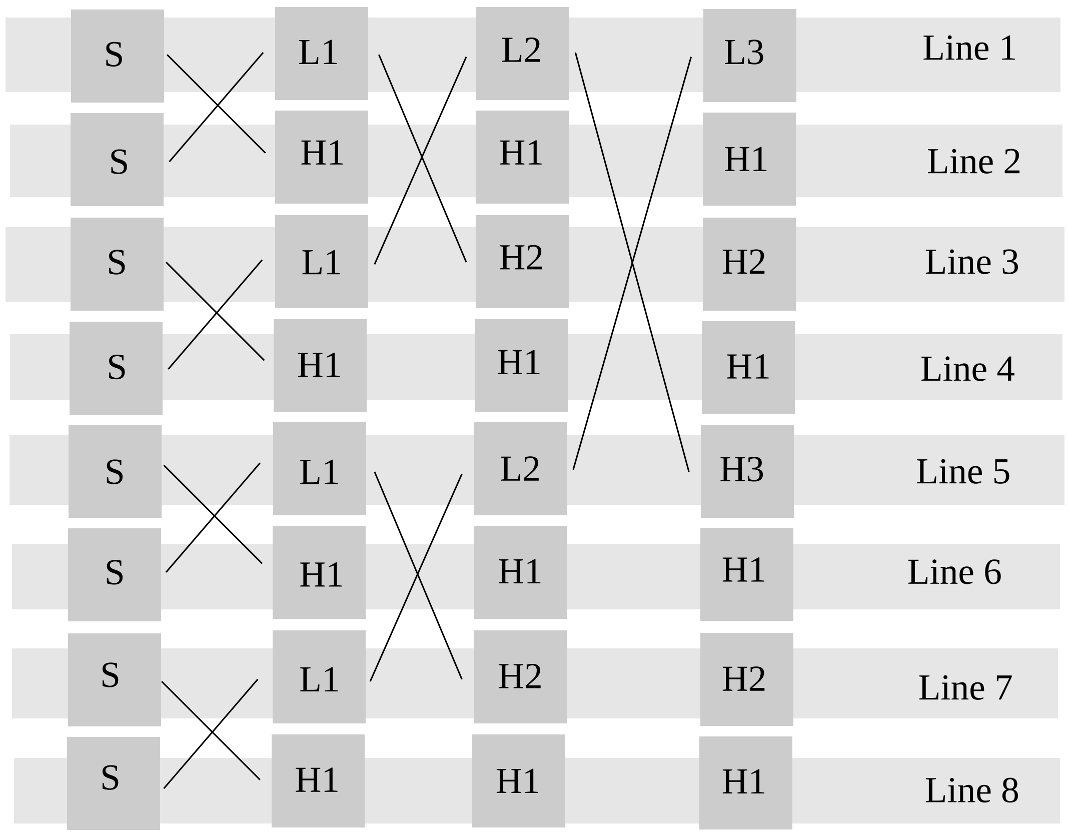

Figure 4.2: Vertical wavelet lifting scheme (Haar)

4.4.2 Horizontal and vertical lifting implementation of Le Gall 5/3 wavelet

In the horizontal lifting scheme, we consider a line of samples. After lifting, even-index samples will be low-pass and odd samples high-pass.

The lifting formula is borrowed from JPEG2000. First, odd samples (high-pass) are computed, then even samples.

$$s_l[2n+1] = s[2n+1] - \left\lfloor \frac{s[2n] + s[2n+2]}{2} \right\rfloor \tag{4.10}$$

$$s_l[2n] = s[2n] + \left\lfloor \frac{s_l[2n-1] + s_l[2n+1] + 2}{4} \right\rfloor \tag{4.11}$$

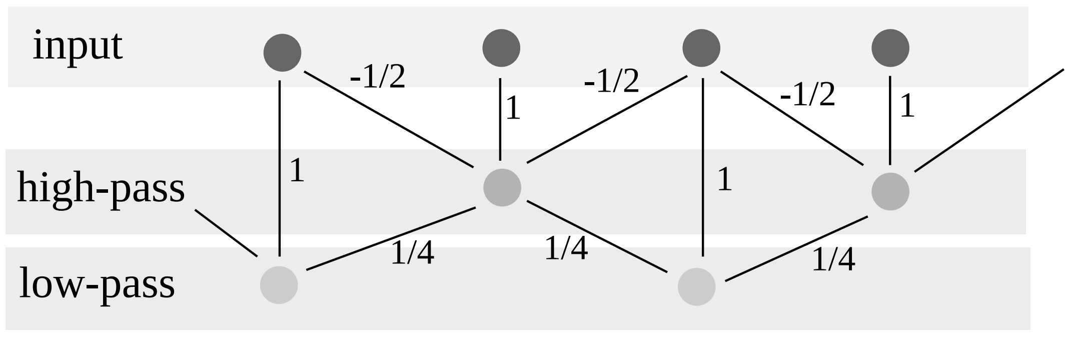

Figure 4.3: Lifting implementation of Le Gall 5/3 wavelet

In the Linelet software implementation, interleaving low-pass and high-pass samples as in figure 4.3 is not a good idea due to cache locality and parallelization concerns. Instead, the two resulting sub-bands are written to separate buffers. The lifting scheme however, still holds as the intermediate results for high-pass band are used to compute subsequent results for low-pass band.

For vertical Le Gall transform, in-place lifting is used, i.e. low-passed and high-passed scanlines are interleaved as in figure 4.2. Contrary to the horizontal transform, interleaving scanlines is not a problem as samples are still tightly packed within a scanline.

In figure 4.4 we see how subbands are split up and organized into a scanline structure.

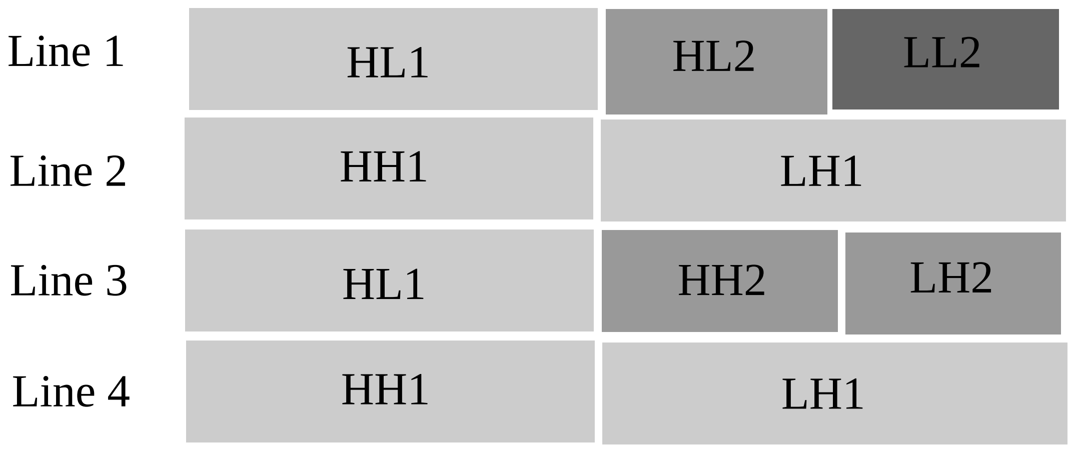

Figure 4.4: Organizing 2 levels of wavelet decompositions in scanlines

4.4.3 Signal extension

Linelet borrows the signal extension used by JPEG2000 (section 2.4.6).

When only two samples are being lifted (vertically), the Le Gall 5/3 lifting step reduces to a simple Haar lifting step due to signal extension.

4.4.4 Combining the vertical and horizontal transform

To get an effective 2D transform, vertical and horizontal transforms must be applied in an interleaving pattern as in figure 4.5.

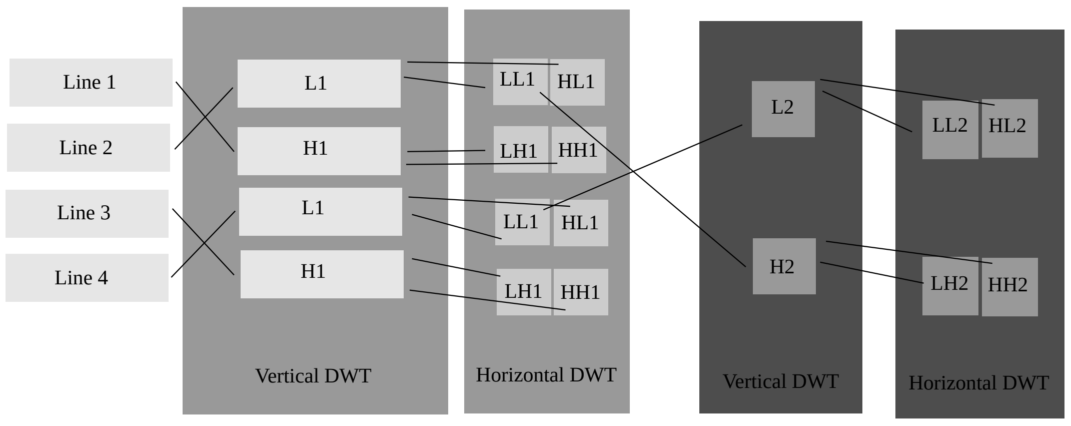

Figure 4.5: 2-level decomposition of 2D DWT in Linelet

If the number of horizontal decompositions is larger than vertical ones, we can simply continue to decompose the LL_n sub-band horizontally as desired.

To ensure the transform is reversible, the order of transformations matter. For simplicity the vertical transform was to be done first, reason being it would reduce code complexity, and allowed faster iteration when experimenting with different vertical transform types.

4.4.5 Numerical precision for lifting

Neither the Le Gall filter nor the Haar wavelet are able to express the output with same dynamic range as the input. With certain input signals, it is possible that the dynamic range of filtered data increases. As the internal numerical precision inside the Linelet encoder is fixed, this gives a maximum bit-depth that is guaranteed to never overflow any arithmetic operation. Proving an exact value is non-trivial with multiple decomposition levels, but 12-bit input was experimentally found to be the maximum possible input bit depth with 16-bit arithmetic.

13-bit input was in artificial edge cases [2] found to rarely overflow operations. If 12-bit is used, neither the RCT nor pre/post filtering (section 4.4.6) can be used as they increase the dynamic range to 13-bit.

4.4.6 A pre/post-filter structure to mitigate slice boundary artefacts

For vertical wavelet transform in Linelet, signal extension at slice boundaries cause a discontinuity between slices. At low bitrates this causes a *striping* artefact, a horizontal-only blocking artifact. With JPEG2000 [16], a similar problem can arise from the use of tiles at lower bitrates, as a discontinuity is introduced, usually called the tiling artefact. For the sake of clarity, *striping* (Linelet), *tiling* (JPEG2000) and *blocking* (DCT) artefacts are essentially synonyms, but refers to different codecs.

While Linelet targets very high bitrates (1:2 to 1:4) where striping effect is not a problem, it was deemed useful to explore techniques to improve visual quality for lower bitrates such as 1:10 to 1:20 as well. Some relevant techniques are enumerated below.

Line-based continuous wavelet transform

Low-latency, blocking-less 2D wavelet transforms can be implemented by employing a line-buffered wavelet transform [38]. The idea is to output data from the wavelet transform as soon as the data required to compute it is available. E.g. with the 5/3 filter, we need to receive scanline $N + 2$ to output the low-pass band for line N. The same requirement holds for every decomposition level. The latency builds up quite fast as each line of latency in decomposition level N corresponds to *two* lines of latency in the $N - 1$ decomposition level. The lowest-frequency basis functions of wavelets are thus very long.

Overall latency must be added equal to the length of the longest basis function. For 3-4 vertical decomposition levels with Le Gall 5/3, the latency would be well over 20 scanlines on top of the number of scanlines already in a slice. The proposal in [38] also requires several tiers with buffering for each decomposition level, which could be complex to implement and difficult to fit into the simple slice structure of Linelet.

Single-Sample Overlap

JPEG2000 Part 2 (Extensions) describes a method, Single-Sample Overlap, which can be used to mitigate the tiling artefact while not requiring the wavelet transform itself to overlap. The idea is to let tiles overlap with one row and column at every subband so that low-pass sample coefficients overlap each other. The cost is a slightly increased file size as overlapping rows and columns have to be redundantly coded. [39] As the ”tiles” in Linelet are very small vertically, sample overlap is likely to add far more overhead than any gain acheived with the method. It is also very likely a patented method.

[2] Fully saturated white noise where each sample was either maximum or minimum of the dynamic range.

Pre/post-processing lifting structure

A method to mitigate this effect - which does not require a scheme like in [38], nor overlapping - was experimented with instead. In the literature, spatial-domain lifting structures similar to what is used in overlapped DCT-based codecs have been attempted for use in wavelet codecs. The pre/post-lifting structure use was borrowed from [5].

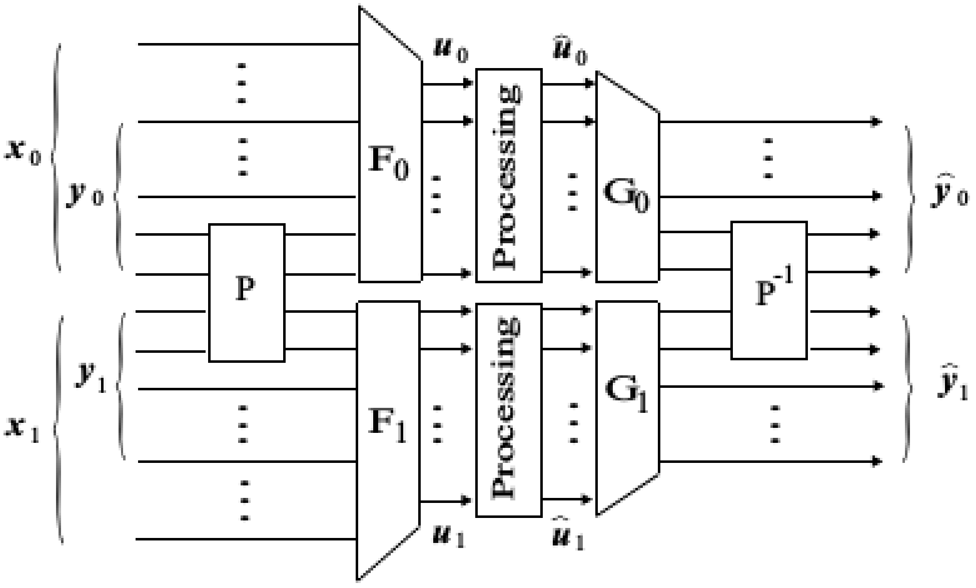

Figure 4.6: Block boundary filter bank model with pre/post-filtering [5]

The purpose of the pre-filter in figure 4.6 is to *introduce* discontinuities, i.e. blocking artifacts, so that the inverse post-filter will effectively apply a deblocking filter.

An attractive property of this scheme is that added latency is fairly low (a few scanlines, or as little as 1 scanline for 2-line filter) as well as being completely modular, i.e. adding this functionality does not require the slice processing to change. It can be optionally enabled at the encoder side as long as the decoder is signaled that post-filtering must be applied after decoding. [5] also suggests that post-filtering can be used even without pre-filtering for deblocking purposes, but this would only be usable as a pure post-processing technique at very low bitrates.

To be fully reversible for lossless compression, a lifting implementation can be used. [5] suggests the use of a *Fast V* lifting structure (figure 4.7).

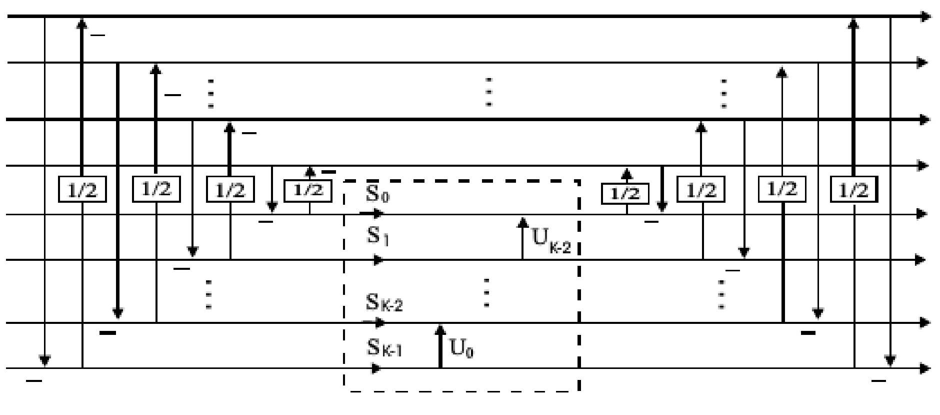

Figure 4.7: Fast V-II prefilter lifting model [5]

2, 4 and 8-line pre/post filters were implemented using the optimal integer coefficients in [5] [3].

Essentially, what figure 4.7 does is extracting high-pass components in the lower scanlines (left butterflies), scaling up the high-passed lines (center **V** matrix), and reversing the low/high-pass extraction (right butterflies). The inverse process would extract the high-passed components, but *reduce* their amplitude instead, achieving the desired low-pass deblocking effect.

Problems with multiple decoding/encoding passes

An important requirement of production codecs is that they must allow multiple generations of encoding without any significant loss in quality.

With the pre/post filtering structure, this is a problem as while the prefilter followed by postfilter is perfectly reversible, the postfilter followed by prefilter is not. The postfilter attenuates high-pass samples, and the quantization error introduced cannot be recovered.

With multiple generations of encoding, the decoded output is fed back into the encoder, which means the postfilter in the decoder is followed directly by the prefilter in the encoder. As mentioned, this is not guaranteed to be reversible, and we will introduce additional errors.

Coding gain of pre/post lifting structure

Pre-filtering could also potentially improve coding gain of the wavelet transform due to a possibility to decorrelate across slices, but this was not found to be the case in Linelets implementation, quite contrary. Despite slightly reduced coding gain for lossless material, pre/post-filtering gave significant gains both PSNR-wise and visually at lower bitrates. To gain an insight as to why this happens, we look at theoretical transform coding gains with and without prefiltering structures using a simple 2-line Haar transform. We also look at reconstructed errors (MSE) when the high-pass band has been quantized entirely to 0, as is common at lower bitrates. Reconstructed errors are calculated with and without pre/post-filtering.

For the purpose of this analysis we assume an image consisting of 4 scanlines with a slice height of 2 (two slices). We assume that vertical samples are correlated with each other and follow an $AR(1)$ model, i.e., the autocorrelation of samples are given by the formula

$$R_{xx}(l) = \rho^{|l|}\sigma_x^2 \tag{4.12}$$

We pick out a column vector of the image $\mathbf{v} = [a, b, c, d]^T$. The variance for each sample is σ_x^2.

For one level of decomposition with the Haar wavelet lifting scheme [4], we obtain

$$\mathbf{v}_h = \left[\frac{a+b}{2}, b-a, \frac{c+d}{2}, d-c\right]^T \tag{4.13}$$

which has variance for resulting coefficients equal to

$$E[\text{diag}\{\mathbf{v}_h\mathbf{v}_h^T\}] = \sigma_x^2\left[\frac{2+2\rho}{4}, 2-2\rho, \frac{2+2\rho}{4}, 2-2\rho\right]^T \tag{4.14}$$

It is important to note that the Haar lifting scheme is a biorthogonal transform, not orthonormal, and for uncorrelated input places more power in high-pass band than low-pass. We use a weighted coding gain formula to compute the sub-band coding gain [40]

$$G_T = \frac{\sigma_x^2}{\sqrt[N]{\Pi_{b=0}^{N-1}\left(G_b\sigma_{Y_b}^2\right)}} \tag{4.15}$$

[3]These coefficients are computed for 9/7 filter only, but seemed to work for 5/3 as well.

[4]For 2 samples and signal extension, Le Gall 5/3 filter reduces to Haar as well.

where sub-band synthesis gains are 2 for low-pass and 0.5 for high-pass band. For Haar transform with a given AR(1) model with ρ, this computes to

$$G_T = \frac{2}{\sqrt[4]{(2+2\rho)^2(2-2\rho)^2}} \tag{4.16}$$

If we assume $\rho = 0.95$ as is common for still image coding analysis, we obtain a coding gain of

$$AR95_{\text{dB}} = 10\log_{10}\left(\frac{2}{\sqrt{3.9 \cdot 0.1}}\right) \approx 5.055\text{dB} \tag{4.17}$$

We now apply a 2-line prefilter structure to the input as in figure 4.7 with scaling factor $S_0 = 2$.

$$\mathbf{v}_p = \left[a, \frac{3b}{2} - \frac{c}{2}, \frac{3c}{2} - \frac{b}{2}, d\right]^T \tag{4.18}$$

and after Haar transform

$$\mathbf{v}_{ph} = \left[\frac{a}{2} + \frac{3b}{4} - \frac{c}{4}, \frac{3b}{2} - \frac{c}{2} - a, \frac{d}{2} + \frac{3c}{4} - \frac{b}{4}, d - \frac{3c}{2} - \frac{b}{2}\right]^T \tag{4.19}$$

Again, we find the variance of resulting coefficients

$$E[\text{diag}\{\mathbf{v}_{ph}\mathbf{v}_{ph}^T\}] = \sigma_x^2\,[A, B, A, B]^T \tag{4.20}$$

where

$$A = \frac{14 + 6\rho - 4\rho^2}{16} \tag{4.21}$$

$$B = \frac{14 - 18\rho + 4\rho^2}{4} \tag{4.22}$$

which gives us a coding gain for $\rho = 0.95$ of

$$G_T = \frac{1}{\sqrt[4]{(2A)^2\left(\frac{B}{2}\right)^2}} = \frac{1}{\sqrt{2.01125 \cdot 0.06375}} \approx 2.79271 \tag{4.23}$$

$$10\log_{10}(2.79271) \approx 4.46\text{dB} \tag{4.24}$$

a result which is 0.5 dB lower than simply transforming without prefiltering. This loss corresponds to roughly 0.08 bit per sample, approximately the lossless coding loss that is observed on real test data.

In this analysis we have not considered that prefiltering the signal increases its power before transform. If prefiltering a signal increases signal power, the postfilter must attenuate the signal, causing an attenuation of quantization errors, hence giving us increased coding gain. We reuse (4.18) and study variance for this signal.

$$E[\text{diag}\{\mathbf{v}_p\mathbf{v}_p^T\}] = \sigma_x^2\left[1, \frac{10-6\rho}{4}, 1, \frac{10-6\rho}{4}\right] \tag{4.25}$$

We average over the vector and obtain a variance of

$$\frac{\sigma_{x_p}^2}{\sigma_x^2} = \frac{14-6\rho}{8} \tag{4.26}$$

which for $\rho = 0.95$ computes to a ratio of 1.0375, or $10\log_{10}(1.0375) \approx 0.16$dB, not enough to compensate for the lower coding gain with prefiltering. In practice, these approximations appear to hold for lossy coding at very high bitrates (1:2 compression rates and above), and makes pre/post-filtering not worthwhile for very-high bitrate coding.

At much lower bitrates however, when high-pass bands are quantized to 0 most of the time, we observe that pre/post filtering begins to improve PSNR. To simulate this low-bitrate case, we zero out the reconstructed pass-band for (4.13), and perform inverse transform.

$$\hat{\mathbf{v}}_h = \left[\frac{a+b}{2}, \frac{a+b}{2}, \frac{c+d}{2}, \frac{c+d}{2}\right]^T \tag{4.27}$$

we take the error vector $\mathbf{e_h} = \hat{\mathbf{v}}_h - \mathbf{v}$ and compute the error

$$E[\mathbf{e}_h^T \mathbf{e}_h] = (2 - 2\rho)\sigma_x^2 \tag{4.28}$$

If we take the pre/post-filtering case however, we reconstruct a vector after inverse Haar transform

$$\hat{\mathbf{v}}_{ph} = \left[\frac{2a+3b-c}{4}, \frac{2a+3b-c}{4}, \frac{2d+3c-b}{4}, \frac{2d+3c-b}{4}\right]^T \tag{4.29}$$

and postfiltered

$$\hat{\mathbf{v}}_{php} = \left[\frac{2a+3b-c}{4}, \frac{6a+8b+2d}{16}, \frac{2a+8c+6d}{16}, \frac{2d+3c-b}{4}\right]^T \tag{4.30}$$

with error vector

$$\mathbf{e}_{php} = \left[\frac{-2a+3b-c}{4}, \frac{6a-8b+2d}{16}, \frac{2a-8c+6d}{16}, \frac{-2d+3c-b}{4}\right]^T \tag{4.31}$$

which computes to an error

$$E[\mathbf{e}_{php}^T \mathbf{e}_{php}] = \sigma_x^2(2A + 2B) \tag{4.32}$$

where

$$A = \frac{14 - 18\rho + 4\rho^2}{16} \tag{4.33}$$

$$B = \frac{104 - 96\rho - 32\rho^2 + 24\rho^3}{256} \tag{4.34}$$

A is MSE for samples a and d, while B is MSE for samples b and c. For $\rho = 0.95$ we obtain $A \approx 0.031875$ and $B \approx 0.0175664$, a very marginal overall improvement over (4.28) with $\rho = 0.95$. The interesting take from this however is that samples closest to the slice boundary (b and c) have a significantly reduced MSE compared to edge samples (a and d). As we increase the image height, almost all scanlines will be affected by the postfilter, and we can assume a quite significant overall gain. In practice for 2-line slices, gains between 1 dB and 1.5 dB have been observed when applying pre/post filtering at 1:4 compression rates for some material.

4.5 Quantizer

The quantizer in Linelet is a simple round-to-nearest quantizer. Distance between quantization levels are always power-of-two to avoid any multiplication and division in the quantizer implementation and to ensure that repeated dequantization and quantization can be done without any additional loss.

The downside of such a coarse quantizer is reduced ability to accurately normalize quantization noise and account for perceptual weighting (section 4.12, 4.13).

A uniform deadzone quantizer, as usually found in wavelet codecs, was not used as it was found to drastically reduce PSNR at same bitrates. This is likely because there is no conventional entropy coder in Linelet, and merging zero runs is not a very useful way to reduce bitrate.

The Linelet quantizer can be expressed in the C programming language as

```
int16_t quantize(int16_t input, unsigned step_size_log2)
{
   unsigned q = step_size_log2;
   if (q == 0)
      return input; // No quantization

   // We can choose 1 << (q - 1) here as well.
   // Then we would round up instead of rounding down when
   // we're quantizing something right between two indices.
   // Round-down is better in Linelet as
   // quantized coefficients
   // are represented with twos-complement,
   // and the range of twos-complement is slightly larger
   // for negative numbers.

   int16_t rounding = (1 << (q - 1)) - 1;

   // Assume right-shift is arithmetic.
   int16_t quantized = (input + rounding) >> q;
   return quantized;
}
```

Dequantizing becomes trivial with round-to-nearest.

```
int16_t dequantize(int16_t quantized, unsigned step_size_log2)
{
   return quantized << step_size_log2;
}
```

With this simple quantizer, we clarify that *quantizing by N bits* means that we quantize such that *step_size_log2* is N. In section 4.12 and section 4.13 we will also refer to quantizer "offsets". This is an offset applied to *step_size_log2*.

4.6 Precincts

After each scanline has been wavelet transformed and quantized, the coefficients for each sub-band in a scanline is divided into precincts, an idea which is borrowed from JPEG2000.

The precinct size is power-of-two for simplicity and the image width must be divisible by this size. The precinct size per subband depends on the decomposition level where low-frequency bands have fewer samples per precinct.

$$\text{subband precinct samples} = \frac{\text{subband width}}{\text{full image component width}} \cdot \text{constant} \tag{4.35}$$

where a constant of 64 or 128 has been found to give best results. This scaling of precinct size ensures that we will always be able to divide a subband into precincts. A precinct size of 128 allows for all the common HD formats and beyond, 1280x720, 1920x1080, 2K, 4K, etc.

For subsampled chroma, i.e. 4:2:2, number of precinct samples is halved for chroma channels. Number of horizontal wavelet decompositions is reduced by one to ensure that (4.35) is always divisible.

The trade-off with using smaller precincts is that bitrate can be significantly reduced, but requires more processing [5]. At very low bitrates (1:40 compression and below), the cost of signaling precincts becomes a very significant overhead, and using larger precincts can be better.

[5]With the current software implementation, this is a very significant overhead.

4.7 Highly simplified entropy coding - bitplane packing

After rearranging the (quantized) subbands into precincts, a very lightweight ”entropy coding” scheme is employed. To be able to scale up to several Gbit/s on PC hardware, coefficients cannot be coded on a bit-by-bit basis like in binary arithmetic encoding. Even in dedicated hardware encoders, arithmetic coding tends to be the bottleneck even at rates like 100-200 Mbit/s, so this must be avoided at all cost to be able to scale to lightly compressed 4K and beyond with cheap hardware.

A per-coefficient variable length code would likely be too slow as well, as encoded bits must still be treated on a bit-by-bit basis, making several Gbit/s throughput and appropriate rate control difficult. With higher bitrates, performance reduces drastically in variable-length codecs, as seen with ultrafast JPEG encoders [41].

The solution proposed by Linelet is to treat all samples in a precinct the same way. First, the *range* of the samples is studied. If all samples in a precinct is denoted by $P = (p_1, p_2, p_3, \ldots, p_n)$, then

$$p_{min} = \min(0, p_1, p_2, p_3, \ldots, p_n) \tag{4.36}$$

$$p_{max} = \max(0, p_1, p_2, p_3, \ldots, p_n) \tag{4.37}$$

We then need to see how many bits we need to represent numbers with this range using twos-complement signed integers. This can be efficiently done by using the *count leading zeroes* primitive usually available as dedicated instructions on PC hardware.

```
// Pseudo-code
bits(minimum, maximum)
{
   if minimum == 0 && maximum == 0
      return 0

   minimum = bitwise_not(minimum)
   value = max(minimum, maximum)

   // Assuming value is 32 bit. Add 1 for sign bit.
   return (32 - count_leading_zeroes(value)) + 1
}
```

If both p_{min} and p_{max} are 0, as can often happen at lower bitrates, we do not have to encode the precinct at all.

Another big gain using this method is that to study the effects of quantization on bitrate for rate control purposes, we only have to look at the quantized values for p_{min} and p_{max} instead of re-scanning every coefficient.

After deciding how many bits are needed to pack the coefficients in a precinct we consider 8 coefficients at the same time. The MSB of the 8 coefficients are extracted and packed into a single byte. We then extract the next bit of the same coefficients, etc, until we reach the LSB. This guaranteed that we can extract one (or multiple bytes even) at a time.

If the precinct has 2 or 4 samples only we can pack 4 bitplanes or 2 bitplanes into a single byte respectively. Precincts with only 1 sample is not supported. Precincts with less than 8 samples are quite inefficient to begin with, and their use should be limited anyways.

Packing like this is trivial to implement in hardware (shuffling bits around), and very efficient on the x86 architecture, which has an SSE instruction *pmovmskb* for doing bitplane extraction (section 4.11.2).

Bitplane unpacking during decode can be implemented very simply as well. In hardware it should just be shuffling bits in place, and on PC architecture we can create a look-up table of 256 entries which expands the 8 bits to 8 bytes. A single shift and OR-instruction can then move the bitplanes into place very efficiently and is completed with a sign extension.

On x86 PC hardware, packing 20+ Gbit/s is easy to achieve with this scheme. It is mostly for this reason that Linelet claims to be an ultra-low complexity video codec.

4.7.1 Comparison to EZW and SPIHT algorithms

Both EZW [33] and SPIHT [34] algorithms have a concept of bitplanes where wavelet coefficients are coded from MSB down to LSB. Both methods use a hierarchical tree structure where higher-frequency subbands become children of lower-frequency subbands. Encoding the wavelet coefficients consists of walking the tree and encoding decisions on a bitplane-by-bitplane basis. Both methods maintain a list of which coefficients are currently deemed *significant*, and which are not.

A bitplane is encoded in two or more steps (passes). In one pass, coefficients which are already significant get coded directly. In another pass, non-significant coefficients are studied and the entropy coder then makes a decision whether or not the coefficient becomes significant. If so, it encodes a sign bit and puts the coefficients in the set of ”significant” ones.

One gain of the tree structure is to be able to signal that all insignificant children of a coefficient remain insignificant. This allows EZW and SPIHT to spend relatively few bits signaling low-amplitude wavelet coefficients located in higher-frequency subbands.

We see that Linelets idea of bitplanes is quite similar. The main differences however are that Linelet makes a decision for significance over *all* coefficients in a precinct and there is no tree structure. If one coefficient becomes significant, so does all other coefficients. We therefore just have to signal at which bitplane all coefficients become significant (section 4.8). One very large coefficient (e.g. at sharp edges) in a precinct is enough to cause many other coefficients (which might have been 0) to become significant.

It is obvious that the proposed bitplane packing cannot approach the theoretical entropy. However, the speed gain in bitplane packing is so immense that it was deemed worthwhile to take a big loss in compression efficiency.

It was found that at 1:2 compression rate, the bit plane packing is just 20-25% above the theoretical entropy [6], and actually gave considerably better compression performance than the exp-golomb variable-length code used by Dirac Pro Low Delay.

As the compression goes below 1:4, it was found that the theoretical entropy approaches 60% of the achieved bitrate. It still remains competitive with exp-golomb.

4.8 Signaling per-precinct bits

As each precinct is packed with a varying number of bits per sample, the number of bits used must be signaled in the bitstream beforehand.

A very simple way to do this is to allocate a fixed number of bits. If the number of bits per precinct is assumed to not exceed 15, we could allocate 4 bits per precinct, making the encoding trivial to implement. At target compression rates 1:2 to 1:4, this usually works out to a fixed bitstream overhead of 1.5-3.0%, which might be acceptable, but at lower bitrates, this fixed overhead becomes a big bottleneck where a very large part of the bitstream is allocated just to signal precinct bits.

To mitigate the problem, a very simple variable length code was implemented. First, it was found that the number of precinct bits was quite correlated across a sub-band. Simple, first order prediction could then be used.

```
// Pseudo-code
encode_precinct_bits(bits, num_precincts)
```

[6] Found by estimating coefficient distributions in the different subbands individually, no context adaptation

```
{
   for i in range(num_precincts)
   {
      // bits[-1] == 0
      if bits[i - 1] > 0
         encode_signed_unary(bits[i] - bits[i - 1])
      else
         encode_unary(bits[i])
   }
}
```

A simple prefix-free variable length code to code the prediction error was created based on the *unary code*. The unary code is quite trivial and was found to get close enough (10% redundancy) to the actual entropy of predicted values:

Input (decimal)	Code (binary)
0	0
1	10
2	110
3	1110
4	11110
5	111110
6	1111110
n	(n ones)0

Table 4.1: Unary code

As the prediction error can be negative, the code was extended to be signed.

Input (decimal)	Code (binary)
0	0
1	100
-1	101
2	1100
-2	1101
n	($\|n\|$ ones)0(sign bit)

Table 4.2: Signed unary code

The encoder chooses the unary code (table 4.1) if the value predicted from is 0 (prediction error cannot be negative) and the signed unary code (table 4.2) if prediction error can be negative. The only difference between the two is the sign bit at the end.

Using this code tended to give a 50% reduction in overhead vs. coding 4 bits per precinct, and even more for lower bitrates.

An important consideration in choosing the (signed) unary code is that it is trivial to compute how many bits it takes to encode an array of symbols, which is very useful for rate control (section 4.9).

```
// Pseudo-code
coding_cost(current, prev)
{
   if prev > 0
   {
      predicted = current - prev
      if predicted != 0
         return abs(predicted) + 2
      else
         return 1
   }
   else
   {
      return current + 1
   }
}

compute_cost(bits, num_precincts)
{
   cost = 0
   for i in range(num_precincts)
      cost += coding_cost(bits[i], bits[i - 1]) // bits[i - 1] == 0
   bytes = (cost + 7) >> 3 // Round up to nearest byte.
   return bytes
}
```

To speed this process up even more, we can compute a 256-entry LUT which covers the cost of all combinations of *prev* and *current*.

4.9 Rate control

An important design consideration is how rate control is handled. To meet latency and bandwidth guarantees, Linelet must be able to degrade the quality to be able to fit a slice into a given buffer size.

A strength of Linelet is that after wavelet transform and subdividing subbands into precincts we can directly compute the required buffer size to pack all coefficients without having to perform entropy coding.

As adding distortion (quantization) reduces bitrate, a naive way to ensure we meet our required target can be implemented as such:

```
// Pseudo-code
rate_control(slice, buffer_size)
{
   quantization_level = 0 // assume 0 is lossless

   required_size, precinct_ranges =
      compute_slice_size(slice, quantization_level)

   while required_size > buffer_size
   {
      quantization_level++
      required_size = compute_slice_size_cached(slice,
         quantization_level, precinct_ranges)
   }
}
```

Due to the bitplane packing approach used, we only need to know the minimum/maximum range of each precinct. This greatly simplifies the rate control, since we can reuse *precinct_ranges* when computing *required_size*. Essentially, we can add more quantization to one subband in the slice, see how many bytes we have saved, and continue iterating like this until the required rate is met. No entropy coding is required to compute the required sizes which greatly improves encoder performance. Iterating like this is guaranteed to complete in finite time, which is useful for real-time implementations.

For variable bit rate rate control, we can simply enforce a fixed quantizer level, and encode the result as-is.

An important rate-distortion problem to solve is finding the optimal subbands to add distortion to. This problem is tackled in section 4.12 and section 4.13.

4.10 Bitstream layout

The fundamental unit of the Linelet bitstream is the slice, which represents N scanlines worth of video information.

There are a certain number of parameters which apply to all slices. To avoid redundancy, we can group several slices together into a full frame, which allows us to specify codec parameters once per frame (or rarer).

As Linelet is intra-only, we only need to concern ourselves with a still-image syntax. A video can be made by simply concatenating images together with meta-data such as frame rates, timestamps, etc.

The bitstream layout here is only meant to give a rough overview how the codec organizes data. It does not serve as a specification for the Linelet codec.

4.10.1 Syntax for a full frame

```
linelet_frame()
{
    header_magic() // LINELET1
    linelet_header()
    for slice in slices
    {
        slice_syntax()
    }
}
```

4.10.2 Linelet header

The linelet header specifies codec parameters such as width, height, bit-depth, color transform, number of wavelet decompositions, pre/post filtering, etc.

In addition to this, it contains a table of offsets which point to all slices in the image. This allows trivial multithreaded decoding. We only really need a table like this if slices are encoded at a variable bitrate. In an ultra-low latency setup, we would use a fixed number of bytes per slice so we implicitly know the offsets and sizes of each slice and this table becomes redundant.

4.10.3 The slice syntax

A slice consists of video components (Y, Cb and Cr), and associated wavelet sub-bands for each component. With vertical wavelet transforms, the highest frequency bands would normally span multiple scanlines, but to keep simplicity, only individual scanlines are considered. Sub-bands which span multiple scanlines are simply split up. This means that the number of sub-bands per scanline is variable, but it remains fixed across different slices.

```
slice_syntax()
{
   for component in components
   {
      for line in component[lines]
      {
          scanline_syntax(component[line][subbands_count])
      }
   }
   checksum()
}
```

4.10.4 The scanline syntax

In a scanline we find quantization levels for all subbands, packed wavelet subbands and a small bitstream to signal how subbands should be unpacked.

```
scanline_syntax()
{
   quantization_levels()
   for subband in subbands[component][line]
   {
      precinct_bits = predictive_signed_unary(subband[width] /
         subband[samples_per_precinct])
      unpacked_subband = packed_subband(precinct_bits)
   }
}
```

Subbands are ordered highest-frequency first. Quantization parameters apply to all sub-bands in a scanline. This structure implies that different quantizer levels can be applied to different parts of a sub-band if the sub-band spans several scanlines.

4.10.5 Checksum

In order to create a more robust system which can tolerate losses when Linelet is transmitted over IP, a checksum was added to the slice syntax in order to be able to detect if a slice stream was corrupted, either via packet losses or reordering which can be a problem with IP/Ethernet based transmission.

We chose an ultra-fast non-cryptographic checksum implementation, xxhash [7], which was fast enough to avoid adding much overhead to the Linelet encoder. xxhash is licensed under the BSD 3-clause license which allows commercial use. The checksum implementation itself can be trivially replaced, as long as it is fast enough. If checksumming is implemented at a higher level (outside Linelet), a checksum here can probably be omitted.

4.11 Processing time performance

During the design of Linelet, optimizing for processing time was considered the most important aspect. Where compromises had to be made, processing time would take priority.

[7]https://code.google.com/p/xxhash/

Performance gains should be acheived in both hardware and software implementations. Certain designs will only be efficient in either software or hardware which should be avoided.

4.11.1 Multithreading

Designing for multithreading is very important as it allows optimal utilization of general purpose processing hardware. The trend today is that additional computational power is added in the form of multiple processor units. Linelet can be trivially multithreaded as all image slices can be processed independently.

4.11.2 SIMD - Single instruction, multiple data

All significant processing in Linelet was designed with SIMD-processing in mind. Most modern CPU hardware contains special vector instructions which allow for one operation (e.g. addition) to be performed on multiple values independently (vector addition).

```
// "Normal" loop to perform 8 additions
for i in 0 to 7
{
   output[i] = a[i] + b[i]
}

// SIMD-like processing
output[0:7] = a[0:7] + b[0:7]
```

For Linelet, this is very useful as all major processing is done on large chunks of data with the exact same processing being done on each data sample.

x86s SSE and ARMs NEON instruction sets for example allow for one instruction to operate on 128-bits of data. Linelets basic unit of computation is a 16-bit integer, which optimally allows us to speed up computation by a factor of 8x as we can process 8 integers per instruction instead of one.

Linelet was fully optimized for MMX2, SSE2, SSSE3 and SSE4.1 instruction sets. Overall, a 4x overall improvement over the C implementation was observed. Most of the individual processing functions gave close to 8x improvement. The rate allocator however, is mostly serial code and could not be parallelized well. There is also significant I/O overhead which cannot be optimized away.

4.11.3 Considering hardware implementations

While no hardware design was made for the Linelet codec during the project, we can claim that Linelet is a very hardware friendly codec. No processing step requires multiplication of any kind, only addition, subtraction and shift operations are needed. This allows very small, cheap and compact hardware.

The memory usage is also fairly limited. Only memory to hold one or two slices at one time is required, which can feasibly be done with on-chip memory rather than requiring use of external memory.

4.11.4 Considering GPGPU application

Using graphics cards as general purpose computation devices have become feasible the last years due to APIs such as OpenCL and CUDA. GPUs have far higher computational throughput than CPUs but getting close to the theoretical throughput of these devices requires very high levels of parallelism.

One problem with a GPGPU implementation is that low levels of latency cannot be feasibly reached with conventional GPUs. Transferring memory between CPU and GPU is quite costly, and optimal throughput would only be possible when the GPU is processing large amounts of data at one time.

Still, moving wavelet, colorspace processing as well as pre/post-filtering to the GPU should be a good way to improve performance as all these operations are trivially parallelized.

4.12 Noise power normalization

When using orthonormal transforms (e.g. DCT), adding distortion to one transformed coefficient corresponds to a distortion in the spatial domain. The mean square error (MSE) of the transformed coefficients corresponds directly to the MSE in the spatial domain assuming that coefficients are uncorrelated. For biorthogonal transforms however, this is not the case. Different gains are applied to different coefficients during inverse transform, and distortion applied to each coefficient will have different degrees of distortion in the spatial domain.

For biorthogonal sub-band coding such as wavelets, individual sub-bands have different gain factors during synthesis and as sub-bands represent different frequency bands, we will end up with a non-flat noise spectrum after synthesis if we are not careful.

In the Linelet case, we need to study the Le Gall 5/3 wavelet and look at the effects of quantization on the overall noise. For noise normalization purposes, we want to find quantization levels such that the noise is as white (i.e. flat spectrum) as possible. Having a way to ensure a flat noise spectrum is a good starting point for further refinements when we might want to use a weighted noise spectrum in psychovisual optimizations (section 4.13).

Matlab code to generate the plots in this section can be found in appendix C.

One decomposition level case

We can first assume the simplest case, where we have one decomposition level and quantization noise has been added to both sub-bands with equal power. For Le Gall 5/3 synthesis we have a low-pass synthesis filter:

$$G_1(z) = 1 + \frac{1}{2}\left(z^{-1} + z^{1}\right) \tag{4.38}$$

and high-pass synthesis filter:

$$G_2(z) = \frac{6}{8} - \frac{2}{8}\left(z^{-1} + z^{1}\right) - \frac{1}{8}\left(z^{-2} + z^{2}\right) \tag{4.39}$$

If we assume that low-pass and high-pass coefficients are all uncorrelated we obtain an overall synthesis filter (and hence noise) response

$$|G(\omega)|^2 = |G_1(\omega)|^2 + |G_2(\omega)|^2 \tag{4.40}$$

whose plot can be seen in figure 4.8.

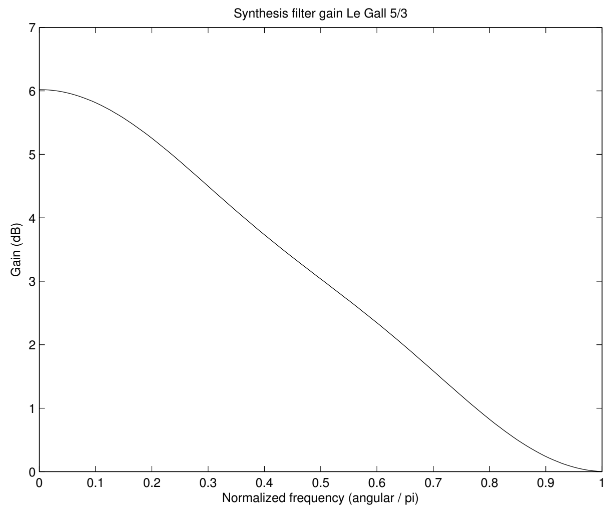

Figure 4.8: Le Gall 5/3 synthesis filter gain

6 dB corresponds to 1 bit. If we quantize the high-pass band by one additional bit, i.e. let the quantizer step size be twice as large as the one for low-pass band, we obtain a noise response which is far more flat than the previous result (figure 4.9).

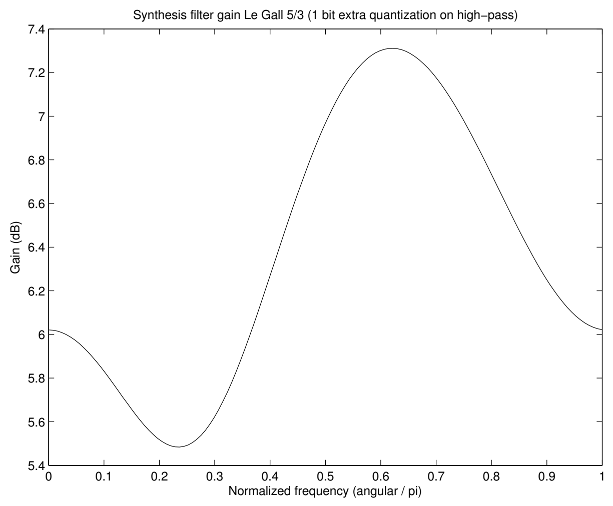

Figure 4.9: Le Gall 5/3 synthesis filter gain with 1 bit additional quantization of high-pass band.

We cannot hope to achieve a perfectly white spectrum, but the approach of adding extra quantization noise to the high-pass band appears to be a good way of flattening the resulting spectrum.

Multiple decomposition level case

For multiple decomposition levels we will move from a mathematical definition to simulated results. We generate gaussian white noise as input signal, apply 5 levels of wavelet decompositions, add white noise (ideal quantization noise) to all sub-bands with same variance and study the final noise spectrum after inverse transform.

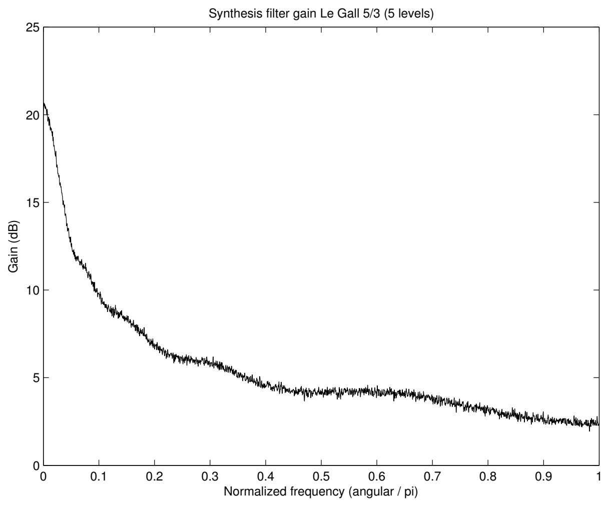

Figure 4.10: Noise spectrum with 5 levels of Le Gall 5/3 decomposition

We now attempt the same scheme as in the single decompostion level case. For each higher frequency sub-band, we add 6 dB (1 bit) of additional noise (table 4.3).

Sub-band	Noise gain (dB)	bits
L_5	0	0
H_5	6	1
H_4	12	2
H_3	18	3
H_2	24	4
H_1	30	5

Table 4.3: Noise gains for figure 4.11

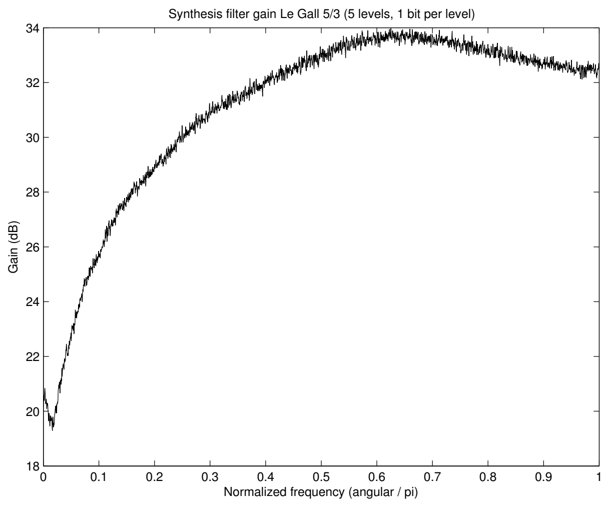

Figure 4.11: Noise spectrum with 5 levels of Le Gall 5/3 decomposition with 1 bit per level compensation.

This is far from flat. To get a reasonably flat spectrum we must take into account that during synthesis of two sub-bands, zero-inserting interpolation is performed which reduces power by half, i.e. 3 dB. Therefore, noise from lower-frequency sub-bands will be implicitly attenuated through multiple synthesis steps. If we use noise gains from table 4.3 and add noise equal to $(3 \cdot \text{synthesis steps})$dB we end up with table 4.4 and figure 4.12.

Sub-band	Noise gain (dB)	bits
L_5	$0 + 5 \cdot 3$	$0 + 5 \cdot 0.5$
H_5	$6 + 5 \cdot 3$	$1 + 5 \cdot 0.5$
H_4	$12 + 4 \cdot 3$	$2 + 4 \cdot 0.5$
H_3	$18 + 3 \cdot 3$	$3 + 3 \cdot 0.5$
H_2	$24 + 2 \cdot 3$	$4 + 2 \cdot 0.5$
H_1	$30 + 1 \cdot 3$	$5 + 1 \cdot 0.5$

Table 4.4: Noise gains for figure 4.12

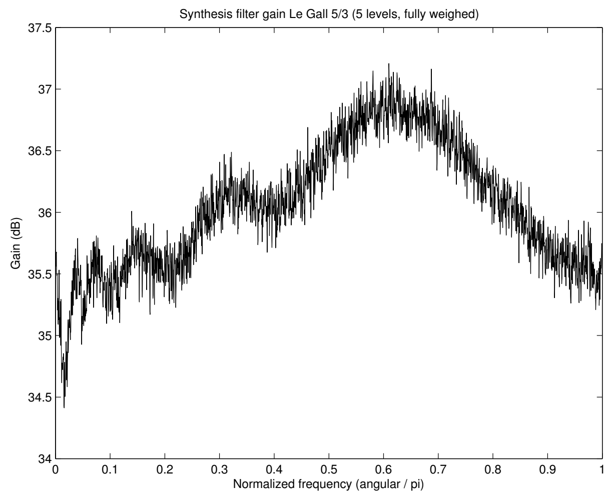

Figure 4.12: Noise spectrum with 5 levels of Le Gall 5/3 decomposition, fully weighted.

This result is almost as flat as the single decomposition level case and can be considered good enough for our purposes.

Two-dimensional decompositions

Extending the previous results to two dimensions is simple. Wavelet transforms are separable and we assume that we can use the appropriate weighting for horizontal and vertical transforms individually and add them [8] to obtain the two-dimensional weighting factor.

In figure 4.13 we see that the low-passed gain is approx 12 dB rather than 6 dB.

[8]Multiplication in the linear domain, but additive in terms of bits and dB.

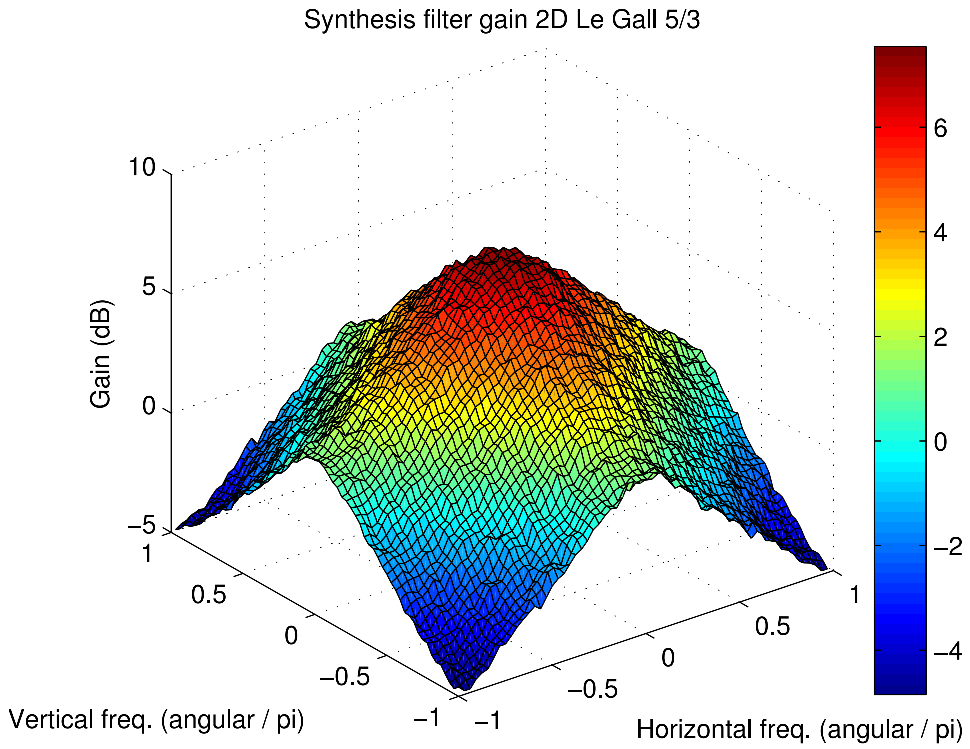

Figure 4.13: Unweighted noise spectrum for 2D Le Gall 5/3 transform.

Using weighting factors of table 4.5 we see the result of weighting in figure 4.14.

Sub-band	Noise gain (dB)	bits
LL_1	0	0
LH_1	6	1
HL_1	6	1
HH_1	12	2

Table 4.5: Noise gains for figure 4.14

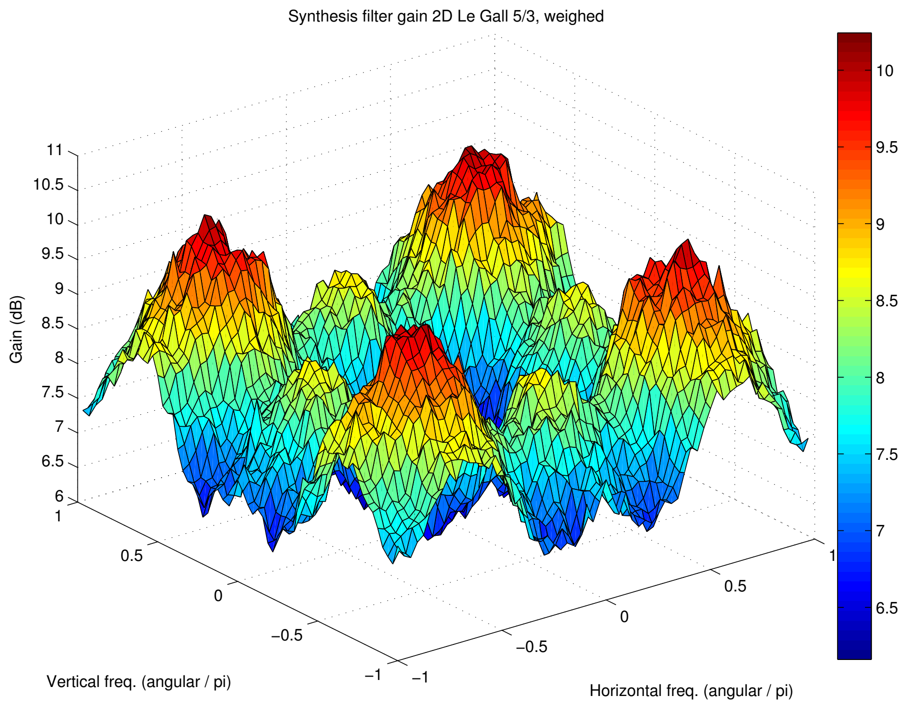

Figure 4.14: Weighted noise spectrum for 2D Le Gall 5/3 transform.

We see a similarity with figure 4.9, except that the top at $\omega \approx 0.6\pi$ is now twice as large in terms of dB.

With 5 levels of 2D decompositions, we apply noise as we would expect from table 4.6. We also account for interpolation attenuation with 6 dB per level instead of 3 dB as we interpolate horizontally and vertically. The result in figure 4.15 is fairly well weighted as we would expect.

Sub-band	Noise gain (dB)	bits
LL_5	0	0
LH_5	6	1
HL_5	6	1
HH_5	12	2
LH_4	12	2
HL_4	12	2
HH_4	18	3
LH_3	18	3
HL_3	18	3
HH_3	24	4
LH_2	24	4
HL_2	24	4
HH_2	30	5
LH_1	30	5
HL_1	30	5
HH_1	36	6

Table 4.6: Weighted noise gains for 5 levels of decomposition

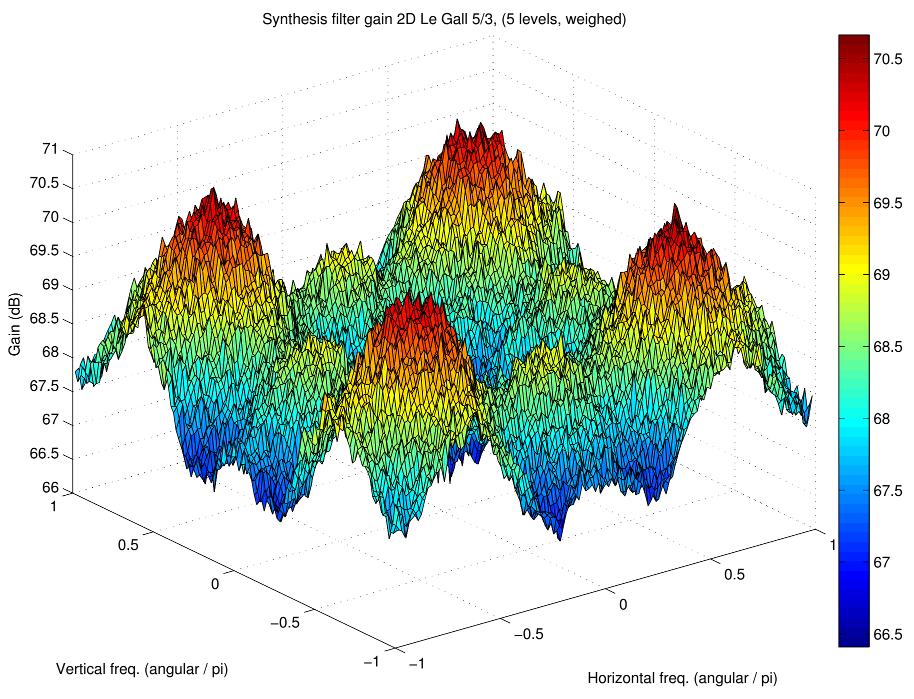

Figure 4.15: Weighted noise spectrum for 2D Le Gall 5/3 with 5 levels of decomposition.

4.13 Psychovisual tuning

The goal is to add distortion in a way that is least noticable to the human visual system. With the sub-band coding approach in Linelet, the only choice for the encoder to make is how much it will quantize the individual sub-bands. As we have multiple color channels (luma/chroma), we will have to evaluate which color channels we are willing to quantize as well.

4.13.1 Contrast sensitivity function

The constrast sensitivity function (CSF) is a function which describes how sensitive the human visual system is to visual patterns at various spatial frequencies measured in cycles per degree (cpd). [42] uses the contrast sensitivity function to optimize JPEG2000 visually and this optimization is used as a basis for visually optimizing Linelet.

[42] implements a non-adaptive (classic) scheme, as well as a spatially-adaptive scheme. When using weighting factors based on frequency, good frequency resolution (DCT) would be preferable, but wavelet transforms have poor frequency resolution at higher frequencies (good time resolution). We therefore need to make quantization decisions for a wide range of frequencies at once, which is not necessarily optimal. [42] overcomes this by being spatially adaptive (utilizing higher time resolution), but such optimization are not practical for an ultra-low complexity codec.

For a classical CSF optimization, we will find a representative cpd for all individual sub-bands, calculate their contrast sensitivities and use the result to compute quantization weighting factors. We can use the center frequency in a sub-band as the representative cpd for the sub-band. This is a very rough assumption, especially at higher frequency sub-bands.

To find cycles per degree, we first find pixels per degree (ppd) for the display device. If we assume a viewing distance v in meters and a screen with a resolution of r dpi we find ppd as

$$f_S = \frac{2v \tan(0.5^\circ) r}{0.0254} \tag{4.41}$$

We assume the video signal is sampled at Nyquist rate, so the maximum cpd observed is half this resolution

$$\text{cpd}_{\text{max}} = 0.5 \cdot f_S \approx 0.34 \cdot v \cdot r \tag{4.42}$$

The contrast sensitivity function itself is not uniquely defined. The CSF must be found empirically and results vary between studies. The luminance CSF plotted in [42] shows a peak in sensitivity at approx. 4 cpd. This non monotonous decrease in sensitivity with frequency is problematic for implementation. [42] mentions that optimizing for a close viewing distance can cause sub-bands other than the LL sub-band to be deemed most important. If LL is coarsely quantized because of this, moving further away from the screen can introduce visible artifacts, as the LL sub-band suddenly becomes most significant. For this reason, we can flatten the CSF to avoid this scenario.

4.13.2 Psychovisually weighted quantization of Le Gall 5/3 wavelet

For sake of simplicity, we just want to find usable quantization weighting factors which work well in most cases. Essentially, we want the quantization noise to be shaped in a way that corresponds well with the inverse sensitivity of the human visual system.

A closed-form function for the CSF plotted in [42] was not found, so a useful luminance CSF definition proposed by Manos and Sakrison [43] was used instead

$$\text{csf}(f) = 2.6\,(0.0192 + 0.114f)\exp\left(-(0.114f)^{1.1}\right) \tag{4.43}$$

where f is spatial frequency in cycles per degree. For two dimensions, we assume that direction of the frequency does not matter, and we use $f = \sqrt{f_x^2 + f_y^2}$.

The CSF is a linear definition, so to find a definition in terms of bits we can quantize by, we invert the CSF and take its base-2 logarithm.

$$B(f) = \log_2\left(\frac{1}{\text{csf}(f)}\right) \tag{4.44}$$

A plot of $B(f)$ is seen in figure 4.16. We see a large peak in sensitivity at $\approx$ 8 cpd. As mentioned, we can simply flatten the CSFs values below this maximum sensitivity to avoid a strong dependency on viewing distance. The number of samples in the lowest-frequency bands are so few that losing some compression efficiency for those samples should be considered negligible.

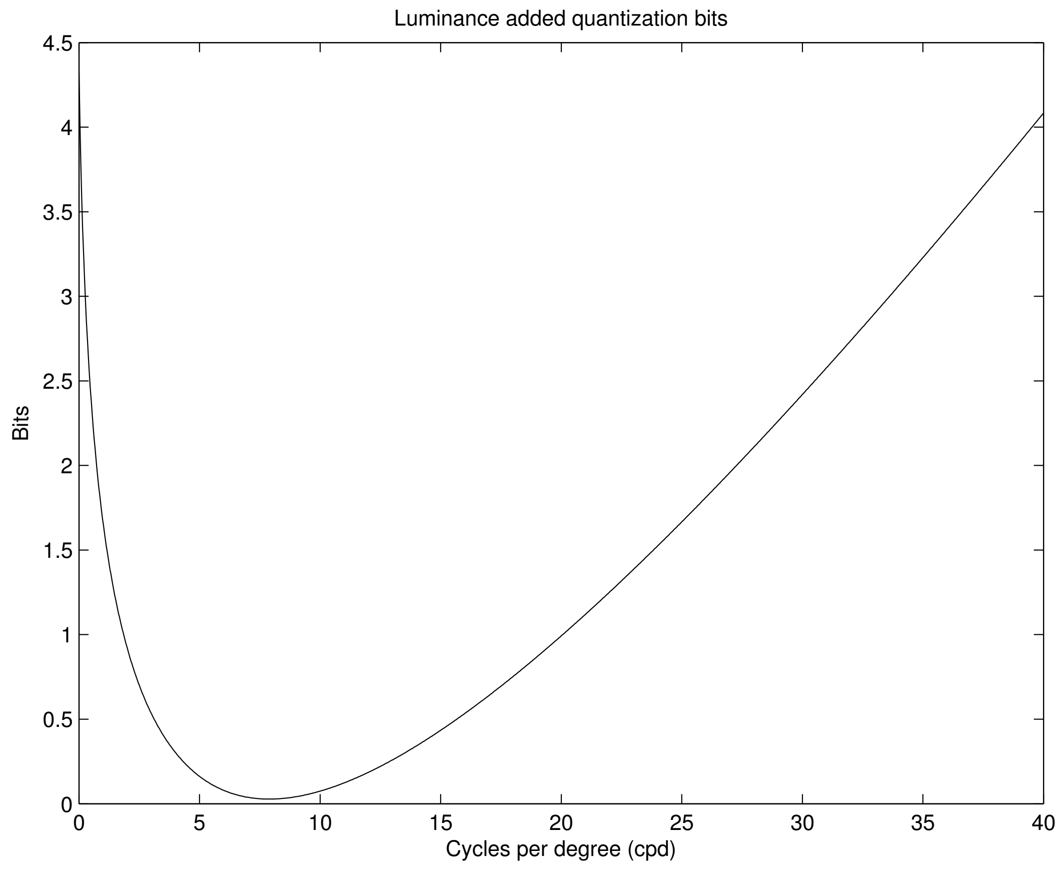

Figure 4.16: Extra quantization bits for spatial frequency.

To find our CSF weighted quantization step size, we can use our normalized quantization parameters from section 4.12 and increase the step sizes based on $B(f)$. We have not considered chroma channels here. We make a simple approximation that we can add one extra bit of quantization to chroma. Ideally, chroma would be quantized using its own CSF, but appropriate data for chroma contrast is not as readily available nor important.

Quantization offsets

If we assume a 96 dpi monitor viewed at a 1.5 meters distance, we obtain a maximum cpd (4.42) of

$$\mathrm{cpd}_{\mathrm{max}} = 0.34 \cdot 96 \cdot 2 \approx 49 \tag{4.45}$$

For 5 levels of decomposition, we compute spatial frequencies for the individual sub-bands by taking the middle frequency horizontally and vertically and compute $f = \sqrt{f_x^2 + f_y^2}$. In table 4.7 we see the offsets we can add to our quantization matrix in table 4.6.

Sub-band	$\text{cpd}/\text{cpd}_{\text{max}}$	bits $B(\text{cpd})$
LL_5	0.0221	1.5756
LH_5, HL_5	0.0494	0.7318
HH_5	0.0663	0.4655
LH_4, HL_4	0.0988	0.1798
HH_4	0.1326	0.0546
LH_3, HL_3	0.1976	0.0625
HH_3	0.2652	0.2587
LH_2, HL_2	0.3953	0.9144
HH_2	0.5303	1.8096
LH_1, HL_1	0.7906	3.8637
HH_1	1.0607	6.2622

Table 4.7: Sub-band spatial frequencies and quantization offsets for a 5-level wavelet decomposition.

Actually using the results from tables generated like this directly turned out to be complicated. The table drastically changes with minor changes in viewing conditions and it's not easy to find a generic result. There is also a problem that Linelet has a very coarse quantizer and fractional bit offsets are not possible.

A very crude approximation to the results generated by the CSF curves is to simply assume that each decomposition level is equivalent to 1 bit. The modified quantization table used in the Linelet implementation is shown in table 4.8. Noise power normalization is also taken into account.

Sub-band	bits
LL_5	0
LH_5	1
HL_5	1
HH_5	2
LH_4	3
HL_4	3
HH_4	4
LH_3	5
HL_3	5
HH_3	6
LH_2	7
HL_2	7
HH_2	8
LH_1	9
HL_1	9
HH_1	10

Table 4.8: Quantization offsets for trivial psychovisual weighing with noise power normalization.

The positive side of such a trivial approximation is that this scheme applies easily to any number of decompositions and any viewing distance despite not being particular accurate. It is also very easy to compute the quantization offset without any lookup-table. For every low-pass filtering operation horizontally or vertically, we simply subtract 1 bit of quantization.

Actually quantizing HH_1 by 10 bits is very unlikely. If we quantize HH_1 by less, we would end up with negative quantization bits for some of the lower sub-bands. Obviously, this means we should simply not quantize these sub-bands at all.

4.13.3 Complexity masking

Complexity masking is a technique often used by block-based DCT encoders which utilize psycho-visual optimizations. The idea of complexity masking is that in segments of images which feature a lot of *visual energy* (i.e. detail), it is harder to notice compression artifacts. Conversely, in segments with little visual energy (flatter regions), loss of visual energy energy is quite noticable. The encoder can study the image on a block-by-block basis and adjust the quantizer accordingly.

Such functionality is not possible in Linelet as quantizers apply to an entire subband. The precinct structure could allow for variable quantizers per precinct, but this has not been explored.

Chapter 5

Method for subjective evaluation

A subjective evaluation of Linelet was carried out at *Café Media* at NTNU using the guidelines of ITU-R BT.500 [26] as a basis for the evaluation. The purpose of the test was to evaluate how experts would react to losses in quality when encoding with Linelet, most importantly the compression rates where losses would start to become visible to the experts.

Two experts well versed in evaluation of codecs, Per Bøhler and Odd-Inge Hillestad, participated in this evaluation. We refer to these as Expert 1 and Expert 2, but which number corresponds to which expert is intentionally obscured.

Experts according to BT.500 have expertise in image artefacts which can occur during the test. They should not, however, have intimate knowledge of the particular system or have been directly involved in the development of the system.

It must be clarified that this test is not really a subjective test as is normally done with BT.500, but rather an expert evaluation.

5.1 Test environment

The test environment of Café Media is certified to comply with BT.500s recommendations. The experts involved deemed the environment representative for evaluating broadcast signals.

Viewing distance was set to 3H, i.e. three times the monitors active display height.

5.1.1 Monitor

The monitor used was a Pioneer PDP-5000EX running over HDMI at 1080p50. The monitor ran in ”game” mode with default settings, without any post-processing or scaling applied. Brightness and contrast settings were calibrated using guidelines from ITU-R BT.814 [44].

5.1.2 Playback system

Arch Linux x86_64 operating system with mpv media player for playback. The playback machine was equipped with a Solid-State Disk drive to be able to stream lossless 1080p50 clips in realtime (190 MB/s maximum throughput).

5.2 Test material

Test material was based on the 4K RGB variant of the *DCI Standard Evaluation Material (StEM) [45]* as well as test sequences *ParkJoy* and *Horse* from SVT and NRK respectively.

The DCI set consists of a series of 16-bit TIFF images at 24 frames per second. It also includes audio, but audio was not included for this evaluation.

Using a 4K source was not practical for this test as no such display device was available, and the bandwidth required for playback would exceed the capabilities of available hardware. The source was downscaled from the original 4096x1714 resolution to 2048x857 4:2:2 10-bit YUV using the ITU-R BT.709 color matrix [6] and then cropped down to 1920x816 to avoid scaling when displayed on 1080p monitors.

The 1080p50 variant of *ParkJoy* was obtained from [46]. The series of still images were converted to 4:2:2 10-bit YUV with ITU-R BT.709 color matrix. *Horse* was obtained directly from external sources and went through the same processing as *ParkJoy*.

5.2.1 Test sequences and bitrates

Three 10 second clips were extracted from DCI StEM to be used during evaluation and training.

dci1 Start frame 06400, 240 frames

dci2 Start frame 10100, 240 frames

dci3 Start frame 04300, 240 frames

parkjoy and **horse** are 10 seconds long (500 frames), and were used as-is.

The uncompressed bitrate for the DCI sequences is 752 Mbit/s [1] and 2 Gbit/s for the 1080p50 material [2]. 8 different bitrates were targeted (table 5.1). These were expected to cover the range of (visually) lossless to extremely annoying.

Bitrate DCI (Mbit/s)	Bitrate ParkJoy/Horse (Mbit/s)	(% of uncompressed)
375	1000	50
260	700	35
188	500	25
150	400	20
112	300	15
75	200	10
38	100	5
19	50	2.5

Table 5.1: Bitrates used for evaluation of DCI sequences

The lossless reference clip was also used as a test clip along with the lossy ones. The mean impairment for the lossless clip should be very close to *100, imperceptible* (section 5.3.2), and can be useful to evaluate the evaluation bias of the experts.

The particular command-lines used for creating the raw YUV source and test sequences can be found in appendix A. Test sequences were encoded with constant bitrate per slice, which represents the worst case for quality at a given bitrate.

[1] $1920 \times 816 \times 20 \times 24$ (10-bit 4:2:2 is 20 bits per pixel)

[2] $1920 \times 1080 \times 20 \times 50$ (10-bit 4:2:2 is 20 bits per pixel)

5.3 Test method

The Double-Stimulus Impairment Scale (DSIS) method from ITU-R BT.500 was used. This test methodology was used for its focus on comparing against a reference source, instead of only evaluating quality in isolation (Single Stimulus) which would not be useful for judging visual losslessness.

The DSIS session according to BT.500 should not last more than 30 minutes. This limits the number of test clips and number of bitrates which can be tested. For this test, the session was extended to 45 minutes. It was decided that this was necessary to get meaningful results as the number of experts was low.

5.3.1 Introduction and training sequence

The experts were introduced to the subjective test, how test material would be presented, which artefacts would be expected to occur and how the subjects were expected to evaluate the material using the grading scale (section 5.3.2).

The sequence **dci3** was used for purposes of training. The experts were shown the range of quality (2.5% to lossless) present in the test session.

After the training session, experts were allowed to ask questions if the test procedure was not clear from the initial explanation.

5.3.2 Grading scale

The grading scale as defined by BT.500 is a five-step scale.

5	imperceptible
4	perceptible, but not annoying
3	slightly annoying
2	annoying
1	very annoying

To improve accuracy and allow for a more fine-grained evaluation, the scale was extended to 100 points (0 to 100), reusing the terminology from BT.500.

100	imperceptible
80	perceptible, but not annoying
60	slightly annoying
40	annoying
20	very annoying
0	(lowest possible score)

The score 100 was clarified to the experts as a score to be used when he was convinced the impairment was imperceptible. 80 would be used when the experts were convinced of a perceptible impairment. Values between two anchor points were to be used for varying degrees of certainty.

5.3.3 Evaluation session

The test sequences were presented to the experts using the Variant II model as declared in the DSIS method. This model displays the same test sequence twice in a row. BT.500 recommends this for video sequences and tests where very small differences are to be evaluated.

A test sequence was presented along with the reference sequence in the following order

1 Reference sequence (10 s)

2 Mid-gray tone (3 s)

3 Test sequence (10 s)

4 Mid-gray tone (3 s)

5 Reference sequence (10 s)

6 Mid-gray tone (3 s)

7 Test sequence (10 s)

8 Mid-gray tone (8 s)

Voting for the result is to happen only in step 8 according to DSIS, although the figures in BT.500 indicate that voting can also happen during steps 5 to 8.

The 8 steps are repeated back to back for all test sequences. One test sequence is roughly one minute long with Variant II, which allows for about 35 test sequences if one allows 10 minutes for introduction, training sequences and questions.

36 test sequences in total were used. **dci1**, **dci2**, **parkjoy** and **horse** were used during evaluation, using the 8 different bitrates plus lossless for each sequence.

The order of the bitrates used for test sequences were psuedo-random and unknown to both experts and us during the test session (double-blind). The reference source used was always guaranteed to change per iteration to avoid problems where experts could lose track of which clip was reference and which was test.

The test results for the first three sequences were not discarded as recommended by BT.500 as the number of available experts was not large enough to justify discarding test data. Discarding test results would also add the possibility of discarding results from the lossless test sequence.

5.3.4 Evaluation schema

During the evaluation session, experts wrote down their assessment on paper. The schema used is found in appendix B.

Chapter 6

Method for objectively evaluating Linelet

6.1 Objective measures of distortion

When objectively evaluating quality, it is necessary to obtain a value representing quality with an algorithm. Two popular alternatives for computing a quality metric are PSNR and SSIM.

6.1.1 PSNR

PSNR (peak signal-to-noise ratio) is the simplest distortion measure which is useful for evaluating image quality.

$$\mathrm{PSNR}_{\mathrm{dB}} = 20\log_{10}\left(\frac{2^{\mathrm{bits}}-1}{\sqrt{\mathrm{MSE}}}\right) \tag{6.1}$$

$$\mathrm{MSE} = \frac{1}{N}\sum_{i=0}^{N-1}\left(x_i - \hat{x}_i\right)^2 \tag{6.2}$$

6.1.2 SS-SSIM

Single-scale structural similiarity is a measure which takes into account more than more pixel at a time and aims to extract a structural quality metric.

$$\mathrm{SSIM} = \frac{(2\mu_x\mu_y + C_1)(2\sigma_{xy} + C_2)}{(\mu_x^2\mu_y^2 + C_1)(\sigma_x^2 + \sigma_y^2 + C_2)} \tag{6.3}$$

where $C_1 = \left(0.01 \cdot \left(2^{\mathrm{bits}} - 1\right)\right)^2$ and $C_2 = \left(0.03 \cdot \left(2^{\mathrm{bits}} - 1\right)\right)^2$ are twiddle factors which avoid division by 0.

The means, covariances and variances are computed on a block-per-block basis for luminance channel only. In this case, we choose 8-by-8 blocks. To get an SSIM for the entire image, we average the SSIM value for several blocks sampled at different positions. We can for example sample all possible block offsets with a stride of 4 pixels horizontally and vertically to ensure that blocks overlap.

SSIM values tend to be very close to 1 and small differences in SSIM amount to large changes in visual quality. To get a value which is easier to compare, we use a dB variant of SSIM.

$$\mathrm{SSIM}_{\mathrm{dB}} = -10\log_{10}\left(1 - \mathrm{SSIM}\right) \tag{6.4}$$

6.1.3 Best objective metric

The best objective metric is the metric which best correlates with results from subjective evaluation. In the literature, SSIM is reported to be a better metric than PSNR [47]. However, a separate study by Rahayu et.al. [48] found that the PSNR metric performs better than SSIM for high quality cinema material. It is noted in [48] that most studies on SSIM tend to test on lower quality and resolutions.

As Linelet targets the upper level of quality (visually lossless), using a PSNR quality metric is justified.

6.2 Testing rate-distortion characteristics with different latencies

Linelet has a configurable amount of latency. 1 to 16 scanlines can be packed in one single slice, and this parameter determines the latency as well as number of vertical wavelet decompositions [1]. The purpose of this test is to see how adding vertical transforms affects the compression rate.

For this test, we use the same *ParkJoy* test sequence as used in the subjective evaluation (chapter 5). We generate PSNR plots for 1, 2, 4, 8 and 16 scanlines latency with and without pre/post filtering. Compression rates range from 3 % (0.6 bpp) to 50 % (10 bpp).

6.3 Testing multiple-generation encoding of Linelet

Codecs used in production must be able to encode and decode several times without significant quality loss.

For this test, we use the same *ParkJoy* test sequence as used in the subjective evaluation (chapter 5). Four test scenarios are used:

- 500 Mbit/s (25 % compression), without pre/post filter, 7 generations
- 500 Mbit/s (25 % compression), with pre/post filter, 7 generations
- 300 Mbit/s (15 % compression), without pre/post filter, 7 generations
- 300 Mbit/s (15 % compression), with pre/post filter, 7 generations

As mentioned in section 4.4.6, multiple generation encoding with pre/post filtering is not fully reversible as postfilter followed by prefilter is not reversible.

It is useful to test different bitrates as while all slice processing in Linelet is fully reversible, encoding errors can cause issues after decoding if decoded samples lie outside the dynamic range for YCbCr. In that case, clipping is necessary and this error will be introduced in subsequent encoding passes. Lower bitrates are more likely to introduce errors which require clipping.

6.3.1 Codec parameters

Codec parameters used for this test are:

- 8 scanlines per slice
- 0 or 8 line pre/post filter
- 5 decomposition levels
- 64 samples per precinct

[1] $\log_2(\text{scanlines})$

Chapter 7

Results

7.1 Subjective evaluation

The number of experts in this test was only 2. The results here should not be treated as a true subjective evaluation, but we can treat this as an expert evaluation.

BT.500 Annex 2 [26] suggests the use of a per-sequence/per-bitrate evaluation. This would only give two data points per test, and therefore completely useless for taking mean and standard deviation. BT.500 also defines a per-bitrate evaluation where data points would be mean and standard deviation over all sequences and all experts. The deviations are likely to come from the differences in the test clips and not necessarily from deviations between experts. To get more data points, we will take the latter approach.

To present the mean opinion scores, the 0-100 score used during the test is re-normalized back to the typical 5-grade impairment scale by dividing by 20 [1].

For each compression rate, a 95 % confidence interval for the mean is given. A confidence interval with so few data points should be used with care. Since only two experts were involved in this test we also present all the individual opinion scores for each test clip which allows us to see trends more clearly (section 7.1.1).

BT.500 suggests using a screening process to eliminate test subjects which often give values far outside the confidence intervals. This step was skipped as eliminating expert evaluations with so few data points was not deemed useful.

A ”hidden reference”, i.e. lossless sequence was used in the subjective evaluation. In figures 7.1, 7.2 and 7.3, the mean opinion scores are reported as 100 % compression rate for the hidden reference test sequence.

A figure for all test sequences together are given along with separate figures for ParkJoy and Horse sequences and the two DCI sequences are presented. In terms of scene complexity, ParkJoy and Horse are arguably far more ”difficult” source material than DCI.

[1]A score of 20 mapped to *very annoying*

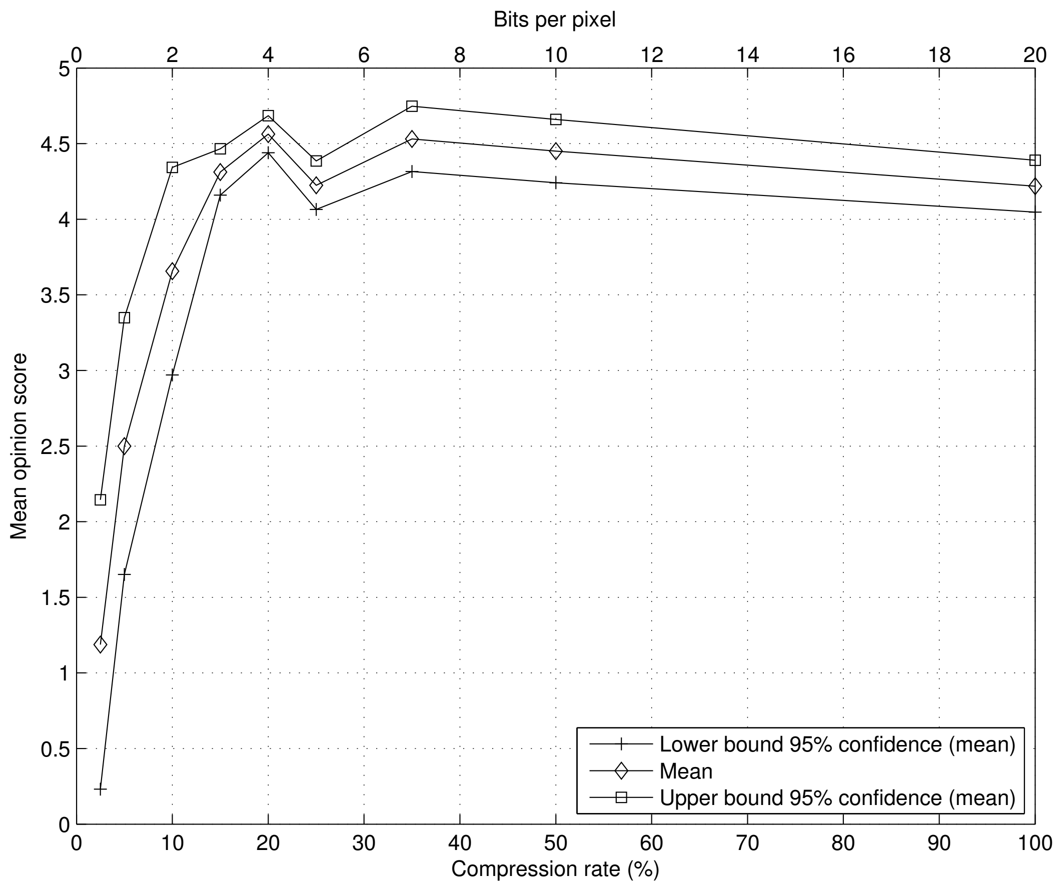

Figure 7.1: Average mean opinion scores for given bitrate over all test sequences and experts

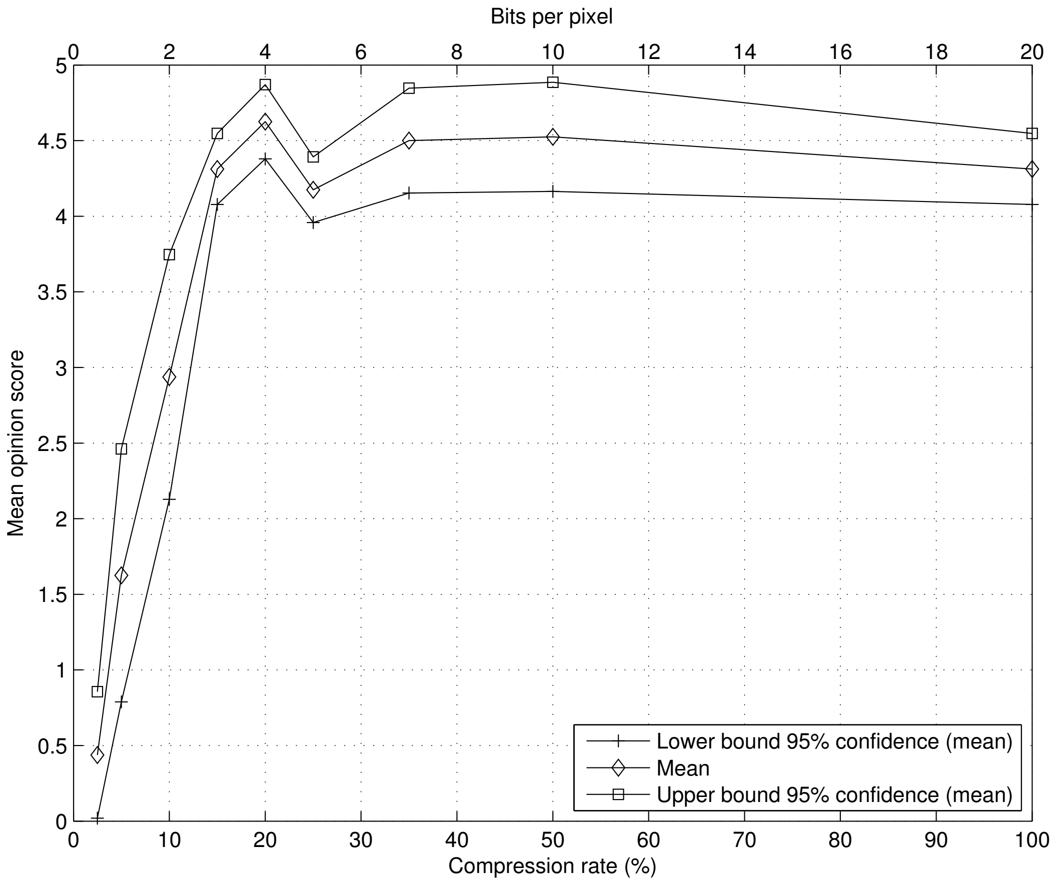

Figure 7.2: Average mean opinion scores for given bitrate over all ParkJoy/Horse sequences and experts

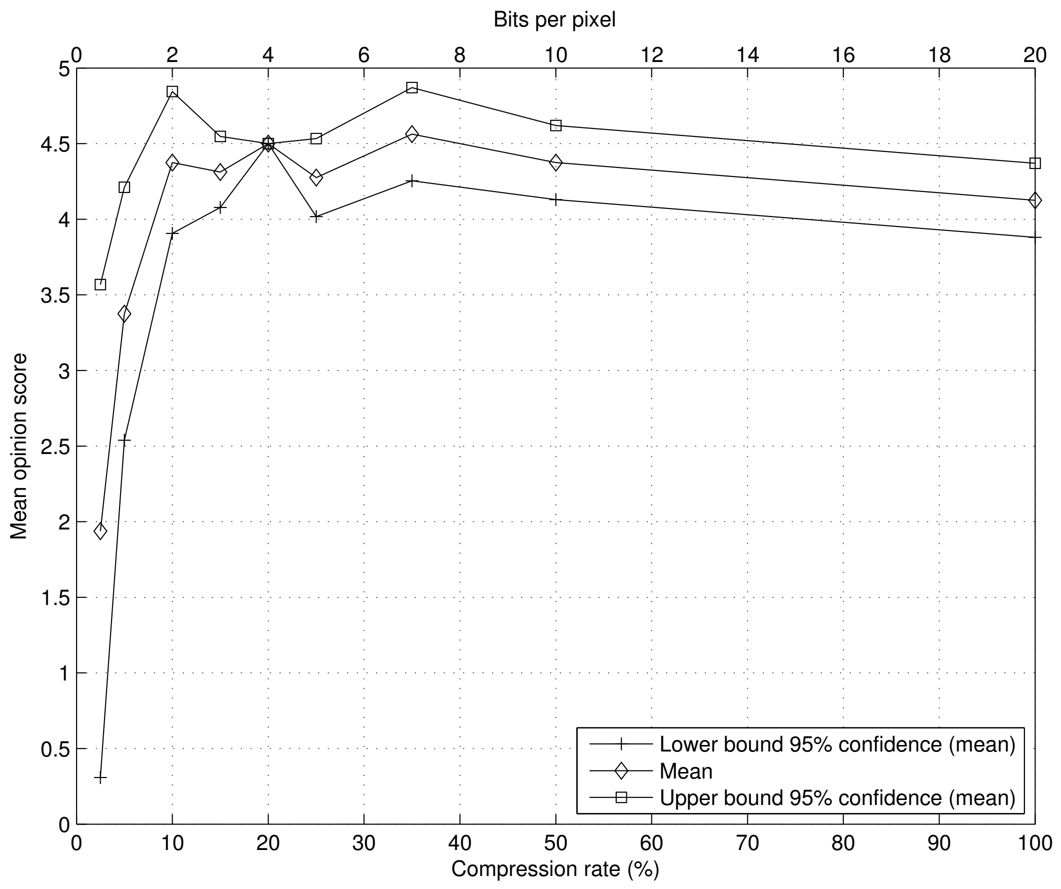

Figure 7.3: Average mean opinion scores for given bitrate over all DCI sequences and experts

From the graphs we see that below 15 % compression, the deviations begin to drastically increase and the mean opinion scores start to drop very fast. Especially for the DCI clips, the lower compression rates 2.5 % and 5 % gave wildly different scores between the experts.

For ParkJoy and Horse, the results are more consistent. Both experts were very familiar with these two test sequences and had an easier time spotting errors in them.

It is interesting to note that the lossless source gave a MOS of slightly above 4 and not close to 5 as expected. This gives a good indication that the expert evaluations were biased in their opinion.

7.1.1 Individual test data

In figure 7.4, 7.5, 7.6 and 7.7 we present the opinion score plots for the experts individually.

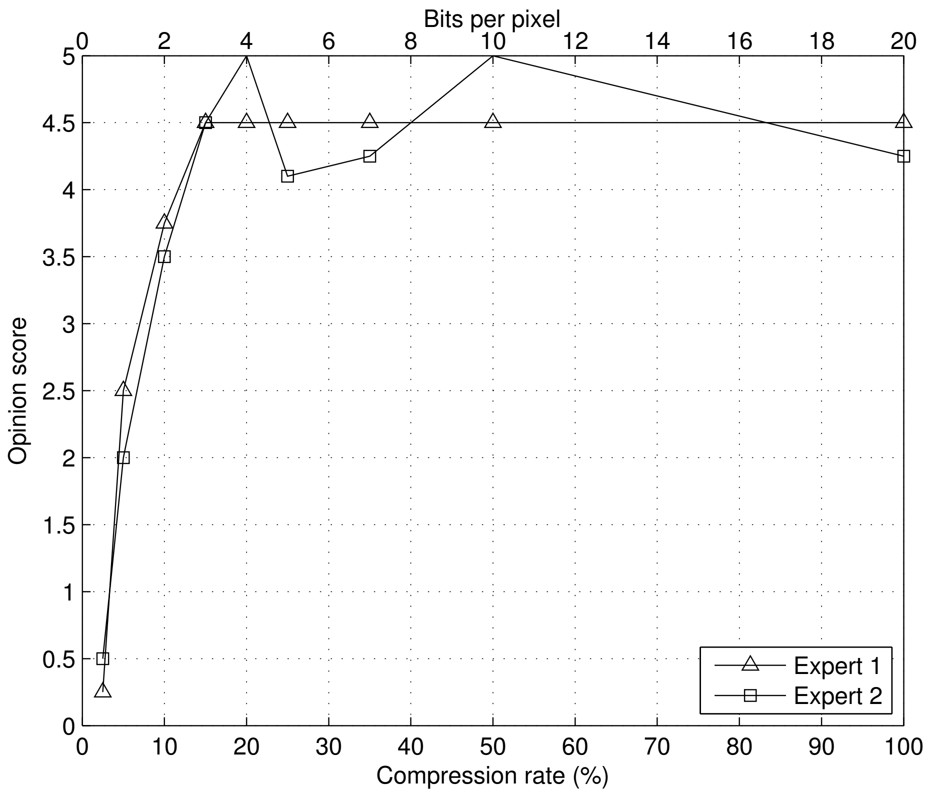

Figure 7.4: Individial opinion scores for ParkJoy

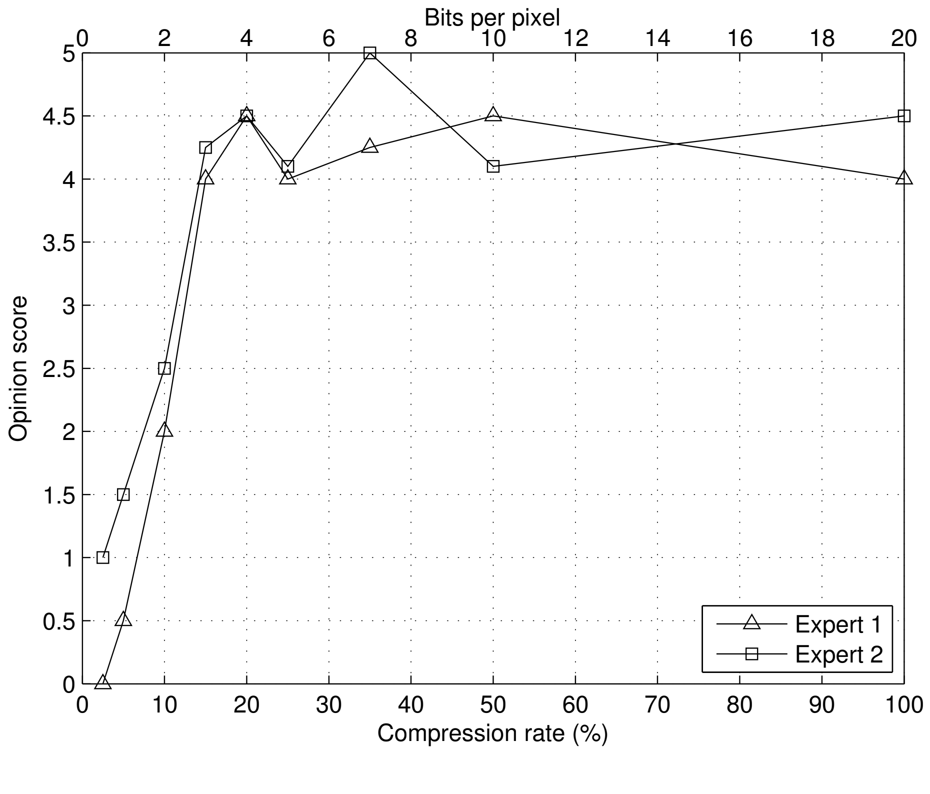

Figure 7.5: Individial opinion scores for Horse

ParkJoy and Horse are well known sequences to the experts and the results are quite consistent between the two. There is no indication that artifacts at 15 % compression is visible to any of the experts.

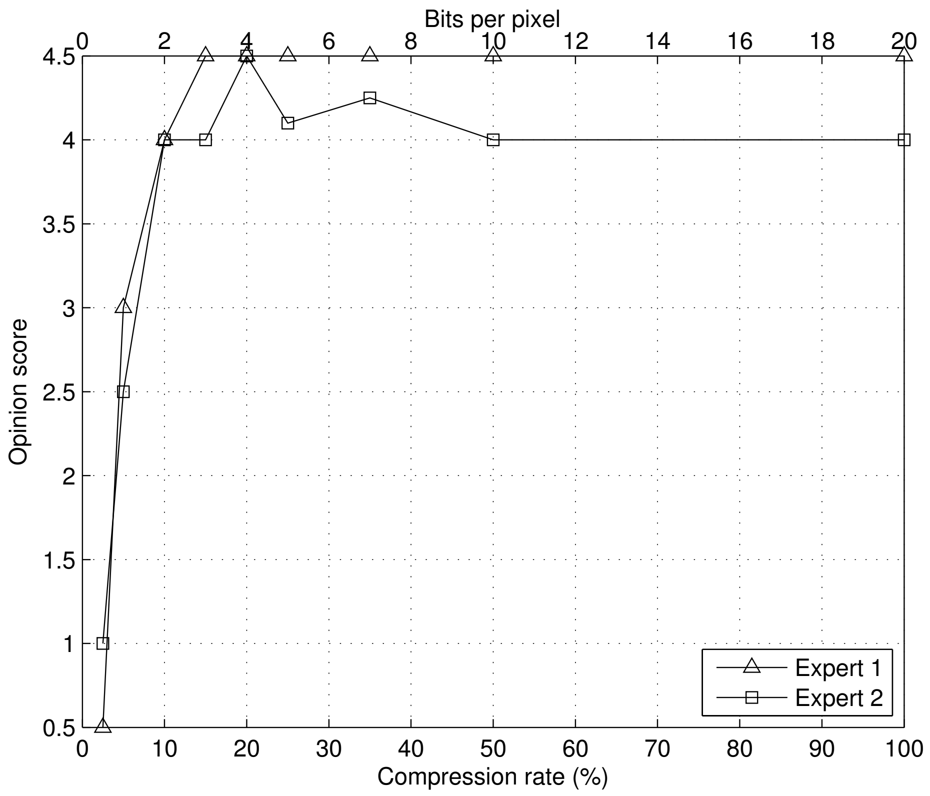

Figure 7.6: Individial opinion scores for DCI clip 1

The results for DCI clip 1 are fairly consistent. Results for expert 1 indicates that 10 % compression is slightly visible, first at 5 % compression do we see an obvious drop in quality.

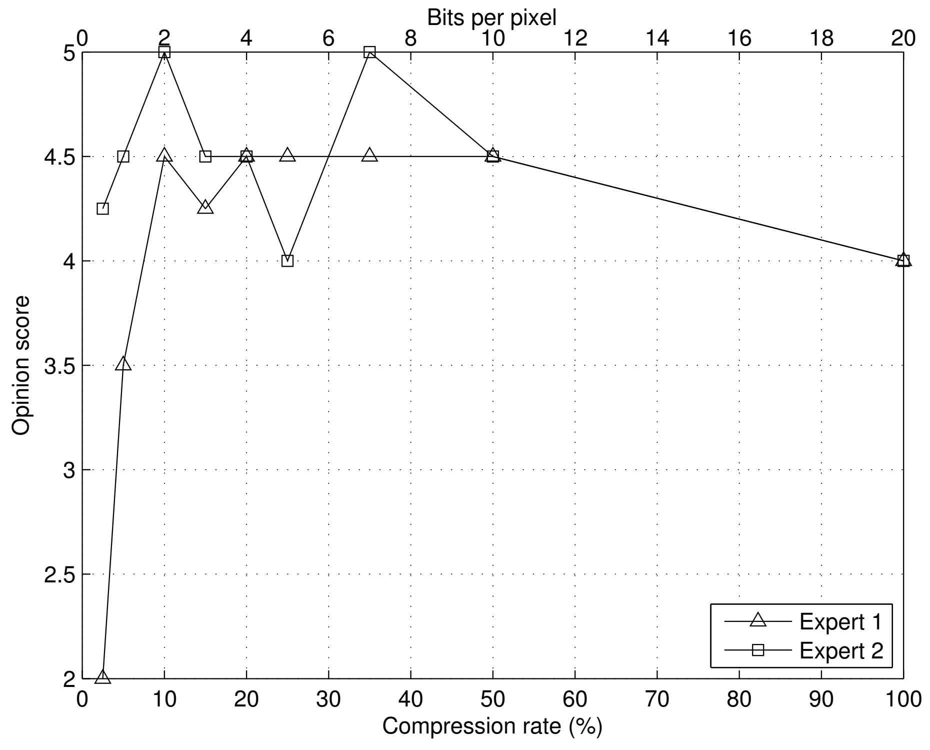

Figure 7.7: Individial opinion scores for DCI clip 2

This test clip turned out to be very difficult to spot any errors in, even at 1:40 compression (2.5 %)

for one of the experts. The results are wildly inconsistent. This is a hint that the test sequence is poor for use in evaluation.

7.1.2 PSNR and SS-SSIM for test sequences at tested bitrates

It was clear that encoding ParkJoy and Horse was far more difficult than the DCI clips. On average, roughly 2 bpp extra (10 % of uncompressed) bitrate was required to obtain same objective quality (SSIM) as the DCI clips (figure 7.8 and figure 7.9).

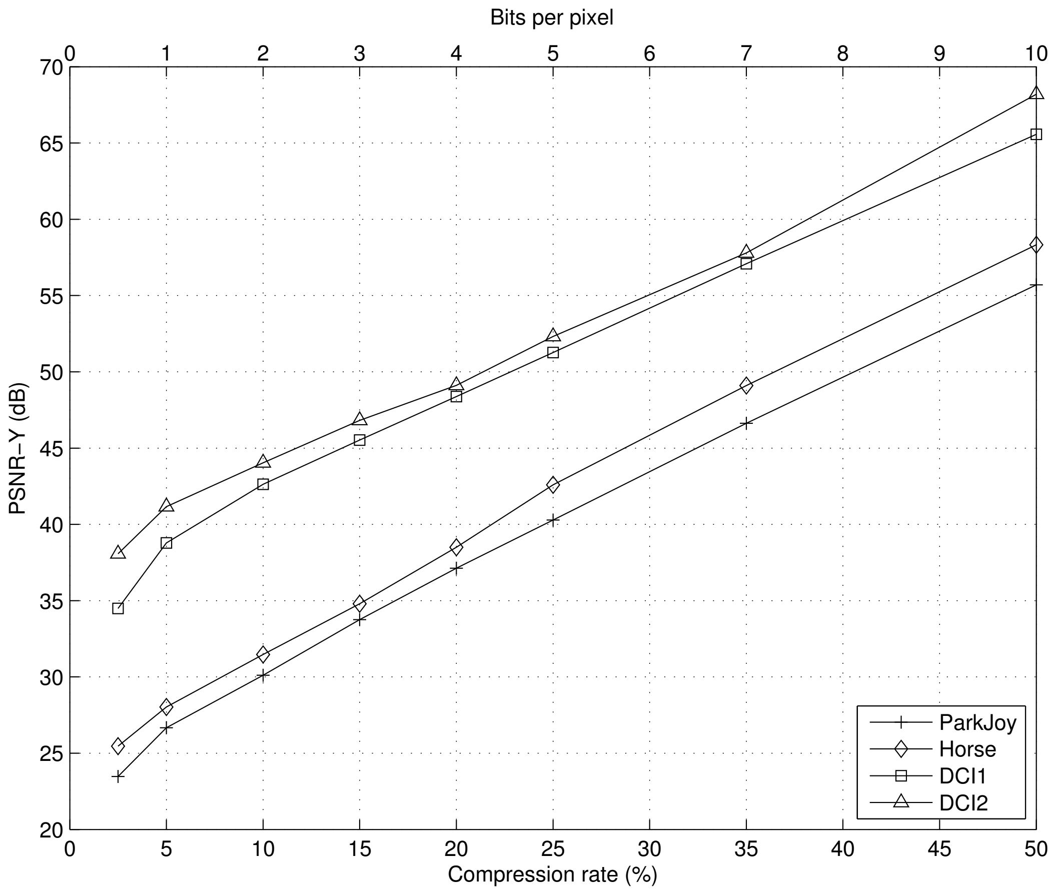

Figure 7.8: PSNR-Y for test sequences at tested bitrates

The DCI clips have very similar objective quality curves, both with PSNR and SSIM metrics. ParkJoy and Horse are similar as well.

An interesting test point is that there is no indication that 1:6 compression of ParkJoy ($\approx$ 15 %) was visible to any of the experts [2], despite the fact that this clip had a very poor PSNR of 33.75 dB. This indicates that Linelet does a good job of ”hiding” the distortions present in this clip.

[2]Both experts gave a score of 90 on the 0 to 100 scale

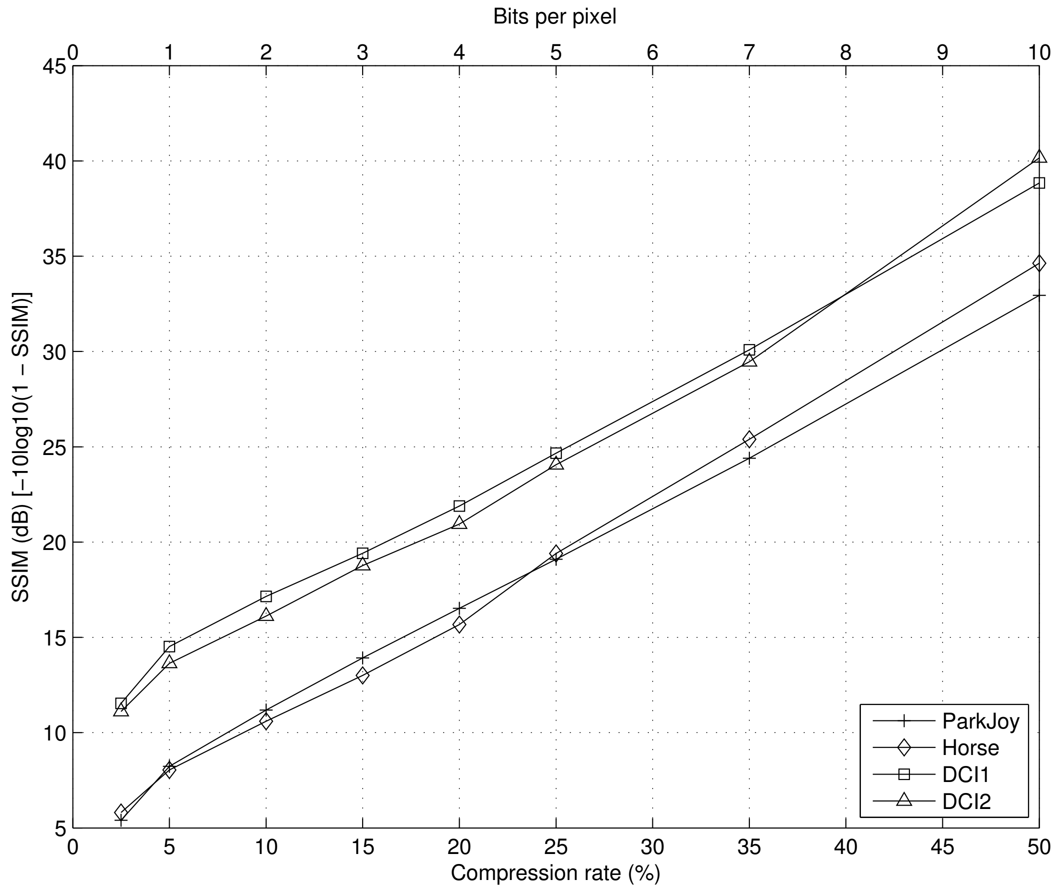

Figure 7.9: SSIM-dB for test sequences at tested bitrates

7.1.3 Global mean opinion score

Global mean opinion score across all sequences and experts was 3.74.

7.1.4 Correlating objective metrics with subjective test data

As the amount of test data is so sparse, it is meaningless to attempt correlating test data with objective metrics. We do however, observe a clear trend in the two expert evaluations. At none of the clips do 15 % compression sequences receive a score which indicate that the experts are able to see any loss when we consider the scores given to the "hidden" reference clip and scores given to sequences with higher bitrates.

7.1.5 Test data

Test data used to generate the results can be found in appendix D.

7.2 Rate-distortion characteristics with different latencies

First, we present the difference in PSNR by using 1 scanline slices and 2 scanlines slices. In figure 7.10 we see a tremendous gain in PSNR at same bitrate, a 3 dB increase. At lower bitrates, the effect of pre/post-filtering on PSNR is noticable.

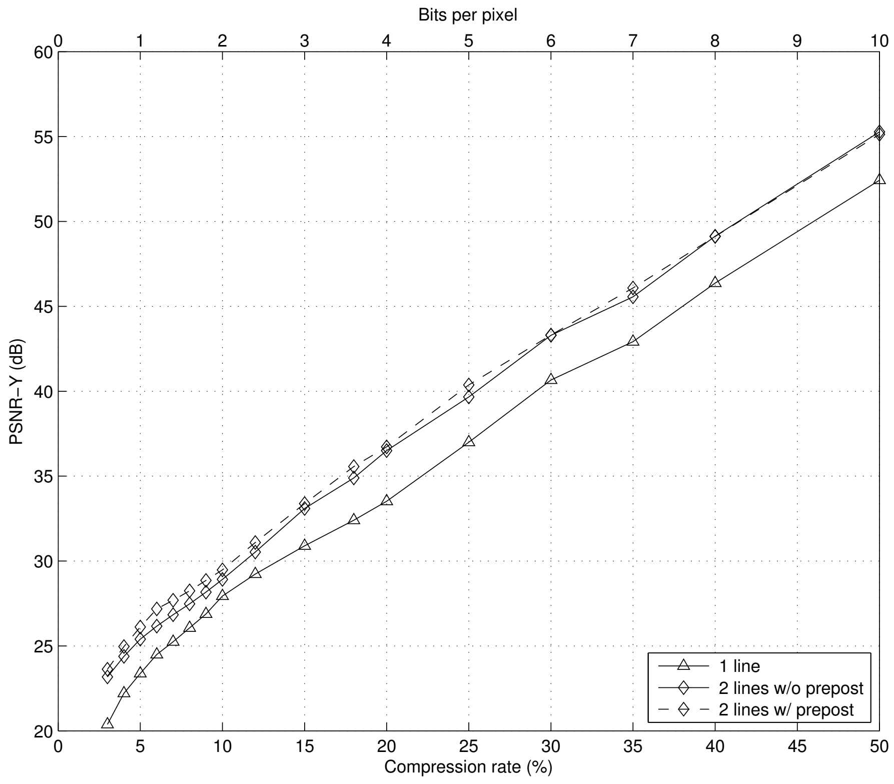

Figure 7.10: ParkJoy rate-distortion with 1 and 2 scanline slices

In figure 7.11 we observe that going from 2 to 4-line slices does not improve the rate-distortion curve significantly at higher bitrates. Below 15 % compression however (see figure 7.11), increasing to 4 lines starts to improve the rate distortion significantly.

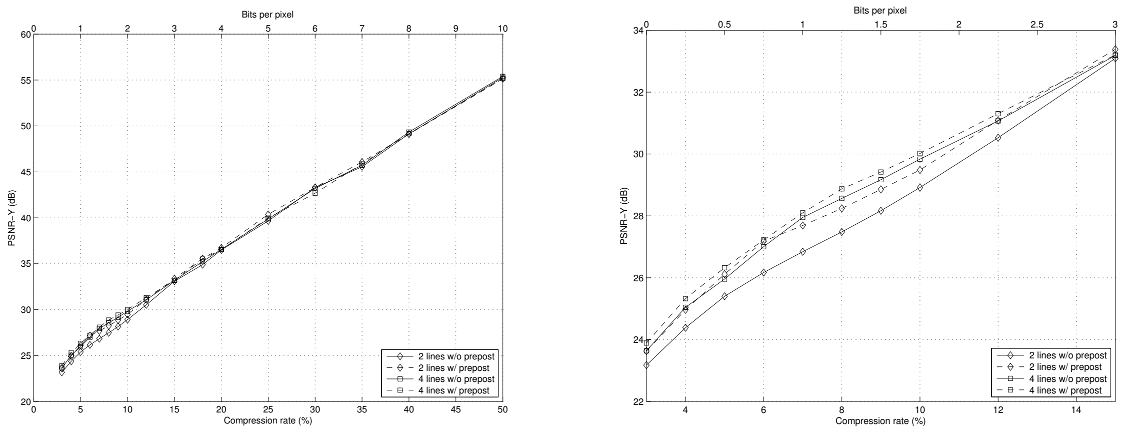

Figure 7.11: ParkJoy rate-distortion with 2 and 4 scanline slices

Beyond 4 lines, diminishing returns start to kick in as we observe in figure 7.12 and figure 7.13. For each decomposition level, a smaller and smaller part of the frequency band is affected.

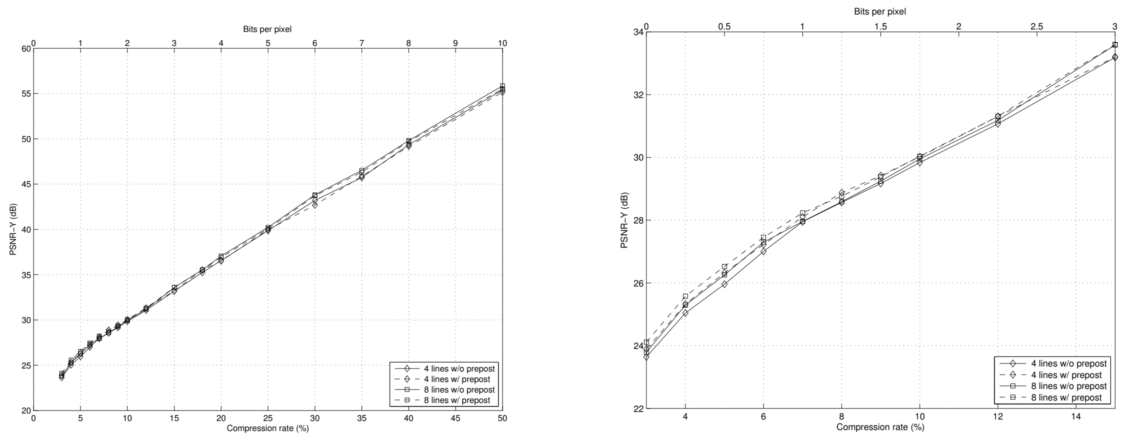

Figure 7.12: ParkJoy rate-distortion with 4 and 8 scanline slices

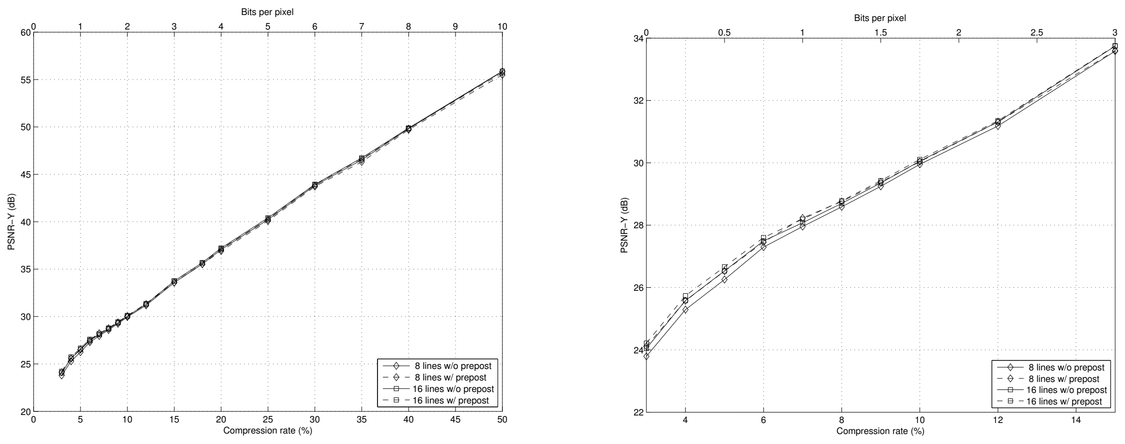

Figure 7.13: ParkJoy rate-distortion with 8 and 16 scanline slices

It is clear that vertical transforms can give great increases in quality at a given bitrate. Just going from 1 to 2 scanlines allows us to reduce bits per pixel by 1.

7.3 Multiple-generation encoding of Linelet

Over 7 generations of encoding and decoding we observe an additional loss of approx. 0.05 dB for pre/post-filtered encodes. After the first encode of 500 Mbit/s parkjoy (figure 7.14), subsequent encodes remain mathematically lossless. While the plot for 300 Mbit/s (figure 7.15) suggests that non-pre/post filtered encodes remain lossless after first generation, per frame studies show occasional additional losses of < 0.0005 dB. This is assumed to be caused by spurious clipping of the decoded signals.

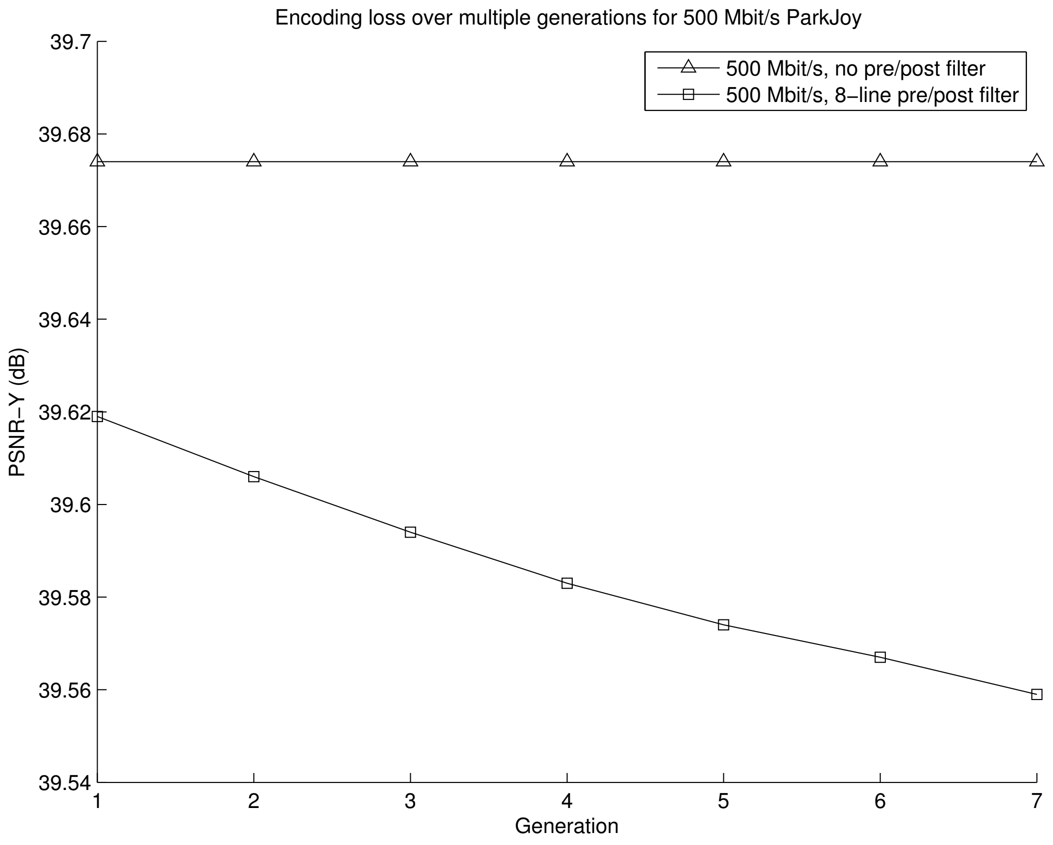

Figure 7.14: Encoding loss over multiple generations for 500 Mbit/s ParkJoy (25 % compression)

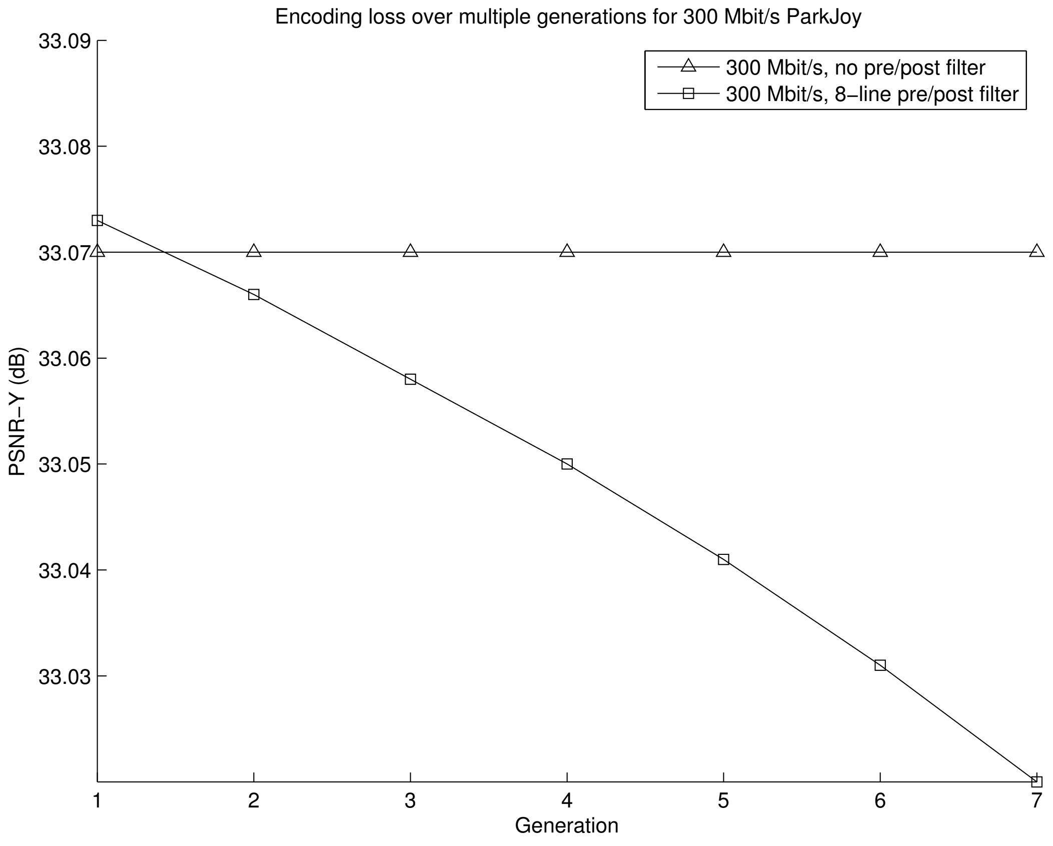

Figure 7.15: Encoding loss over multiple generations for 300 Mbit/s ParkJoy (15 % compression)

This indicates that multiple generation loss with Linelet is mostly insignificant. It is very important to note that this result is only valid when we assume that the exact same Linelet encoder with same parameters is used. The reason we can achieve a mathematically lossless encoding after first generation is that the rate control will deduce all quantization levels used by the first generation. Since all operations are reversible, there is zero additional loss. Dequantization in decoding followed by the same quantization in encoding is lossless in Linelet.

Chapter 8

Discussion

8.1 The choice of few experts against many non-expert test subjects in subjective evaluation

Linelet is designed to be used in production. Linelet is not assumed to ever be encoded or decoded outside a highly controlled production environment. It is not relevant to test with consumers as they will never see such material anyways.

It is far more valuable to test with a few experts which really know and understand the requirements for production-quality material. Relying on expert evaluation however means that BT.500s requirements for subjective evaluation could not be met, which dictates a certain number of test subjects.

If testing with consumers is to be useful, one would have to test with a complete encoding chain, from production codecs all the way down to distribution codecs. Consumers would then evaluate how adding Linelet compression affects the quality of the final distribution stream. Even so, it is very unlikely that any useful test data could be obtained as Linelet operates at distortion levels way below that of distribution codecs.

8.2 Can we be sure that Linelet fulfills requirements for visually losslessness?

In the expert evaluation, two clips were used, ParkJoy and Horse which are very well known in the industry. Especially ParkJoy is very often used in critical codec evaluations due to its very high scene complexity. The use of DCI test sequences however was criticized for being too difficult to find compression artifacts in. The sequences were too out-of-focus and blurry to be useful in subjective evaluation. Test clip **dci2** was considered very difficult to find errors in due to the darker scene along with heavy rain which obscured compression artifacts if one did not know exactly where to look for artifacts.

Even though only 1:2 to 1:4 compression rates are the target for Linelet, not even 1:6 compression was visible to the experts for ParkJoy. We consider it unlikely that 1:4 would be visually lossy for any sensible material at a sensible viewing distance. More expert evaluations and different kinds of tests are required to verify this.

One of the experts noted that Linelet did surprisingly well on ParkJoy and that most codecs fall apart very quickly on this test clip. Most of the typical image artifacts which tend to affect ParkJoy were avoided.

There are of course other kinds of difficult test material and such an important desicion should not be based on two test clips alone. CGI material was not tested for example. CGI material often has very sharp lines and contours which is difficult for transform-based codecs to handle gracefully. Animation (cartoons) is also a similar case. Handling sequences with lots of text is also a difficult case to consider. It is probably worthwhile to test Linelet extensively with these kinds of test material as well.

8.2.1 Other methods to determine visually losslessness

A more difficult test to pass for visually losslessness could be performed by using a split screen view, seeing the reference and lossy version side-by-side for direct comparison. The splitting point could be changed at will to easily compare certain regions of the video.

Swapping out still images (reference and test) at will is also a popular method of determining visual losslessness. However, this method might not be too useful for video as it is not impossible that certain artifacts can be masked when viewed at a high framerate.

For subjective testing purposes in this project, using the DSIS method in BT.500 was deemed the most useful as it is well known, and ultimately, the material will be viewed at a certain distance without the reference source anyways.

8.3 Work left to be done in Linelet

Linelet as it stands is a complete codec. Its decoding could in theory be written as a specification and implemented independently by a third party.

There are however some codec designs which are left to consider before freezing the specification. Most of these designs are complexity and performance tradeoffs. If Linelet is implemented in hardware, we assume that some issues will come up which might lead to a desire of altering the specification slightly to meet hardware demands.

8.3.1 Hardware implementation concerns

While Linelet is designed for easy implementation in hardware, the design has not been verified thoroughly with hardware implementation experts. It is possible that some seemingly efficient designs would be difficult and/or expensive to implement in hardware.

8.3.2 The still-image syntax

Currently, Linelet specifies a still-image syntax which can encode a still image to disk. This still image consists of a header with codec parameters as well as a table of offsets to slices for easy multithreading. After the header, a sequence of Linelet slices are found which represent the entire image top-to-bottom.

For a real, ultra-low latency implementation in hardware, such a syntax is likely ”strange”. A real ultra low-latency system will just transmit Linelet slices, one after the other with fixed bitrate. One only needs a way of configuring a decoder so that it knows certain codec parameters. These parameters are the same for every slice (and frames in a video), and there is no need to transmit redundant configuration data for every single slice.

True real-time operation is not likely to work in practice if codec parameters could change every slice anyways as memory allocation or recomputation of tables is potentially required. It is assumed that these considerations will be resolved if Linelet is implemented in hardware.

At any rate, the ”core” element of Linelet is the slice syntax. It is not expected that this has to change much. How slices are grouped together and managed on the other hand is what is left to decide.

8.3.3 Vertical wavelet transforms

Currently, Linelet employs a vertical wavelet transform. To keep latency and memory consumption low, the image is subdivided into smaller ”slices” which are then encoded independently.

The slice structure is very easy to work with. Slices can be trivially parallelized, latencies are easily computed and fixed buffer sizes can be allocated in hardware.

This study introduced a novel and optional pre/post filtering structure between slices in an attempt to reduce compression artifacts at lower bitrates. This adds a slight latency cost as we need to receive some additional scanlines before we can start encoding a slice.

It is possible that the pre/post filtering structure can be replaced with a continuous wavelet transform. We can still keep our slice structure as-is. However, there will be a tighter dependence between slices as the wavelet coefficients will depend on wavelet coefficients in neighboring slices. With pre/-post filtering, we apply the filters outside the realm of the wavelet transform, but if we were to use continuous transforms, we would have to use a more sophisticated vertical transform implementation.

If continuous wavelet transforms are settled on, the latency will slightly increase. In the slice method, signal extension at slice boundaries must be used which limits the possible latency. With continuous transforms, the encoder must wait until the longest wavelet basis function has died out before the line can be transmitted. In order to get same latency for pre/post as continuous transforms, the number of scanlines in a slice might have to be reduced (i.e. vertical decompositions) to get equivalent latency.

This project has not attempted to implement a continuous wavelet scheme in Linelet, but it is definitely a possibility to explore as it would solve the slice boundary artifact gracefully and possibly improve rate distortion significantly.

8.3.4 Can Linelet be simplified further?

Linelets requirements are 1:2 to 1:4 compression while being visually lossless. It might very well be the case that Linelet has more complexity than what is really needed to achieve this goal considering the very favorable expert evaluation. If Linelet is to be simplified, pre/post filtering support can be dropped as well as reducing the possible number of vertical wavelet transforms. Variable length code of per-precinct bit signalling could also be simplified.

8.3.5 Entropy coding

Linelets entropy coding is intentionally very primitive and simple in order to scale to 5 Gbit/s+ throughput on desktop PC equipment. We believe however, that if Linelet is to make drastic improvements in visual quality, the entropy coding would be the first to change. In informal tests, it was found that bitrate can potentially be dramatically reduced with proper entropy coding.

However, by going down this route one of the great strengths of Linelet is removed, deterministic rate-control and super-high encoder throughput. Entropy coding throughput is a very real challenge in high-end codecs.

A potential improvement is not splitting up sub-bands over scanlines. Instead of having long horizontal precincts, it might be better to use more compact square precincts. This has not been experimented with. If this is implemented however, the slice syntax would obviously have to change significantly.

We believe it is unlikely that the entropy coding of Linelet can be significantly improved without sacrificing speed, especially on PC hardware.

8.3.6 Alternatives to wavelet transform

During this project the alternative of using a different kind of transform has not been considered. The wavelet transform is certainly a very popular transform these days in the production quality bracket of video and image compression. Digital Cinema uses JPEG2000, broadcast contribution is moving towards JPEG2000 and lots of production equipment shoots directly to JPEG2000. Both Dirac Pro and TICO are based on wavelets. In this quality bracket, the DCT appears to be rarely used.

There are several factors which can explain why the DCT is not very popular in this space. First, it is obvious that a ”classic” block-based JPEG-like codec would not work. Having any kind of blocking is very dubious. Modern distribution codecs work around this by employing an adaptive deblocking filter. However, this step is not reversible. This means we would likely see a quite severe multi-generation loss which is not acceptable in production. JPEG2000 after all is well known to have far better multi-generation loss over MPEGs DCT codecs [32].

Even if deblocking is not acceptable, one can still look to a lapped-DCT approach as is considered in the new Daala research codec from xiph.org. Here, a pre/post filtering structure is used, which is quite similar to the pre/post filter in Linelet. This project did not have time to explore if a lapped-DCT approach would make sense for Linelet.

A final concern with DCT codecs is how entropy coding is performed. By far, the normal compression method for DCT is a block-by-block compression scheme where the DCT block is serialized into a vector (swizzling), with some adaptive encoding scheme which exploits the fact that most coefficients will be 0 after quantization. This approach can be problematic for performance however. Wavelets produce higher frequency subbands which consists of many samples. Working with a large number of samples at a time is easy to parallelize and easy to encode. The ultra-fast entropy coding method of Linelet heavily exploits this. With DCT, this kind of optimization cannot be easily exploited.

Finally, wavelets allows trivially varying the number of vertical lines in a slice. With DCT, many different transforms would have to be implemented, for example 4x4, 8x8 and 16x16. For smaller slices, perhaps a 8x2 or 16x1 transform would also be necessary to exploit enough redundancy.

With wavelets, the same 5/3 filter can be used over and over. The only thing to consider is how many decompositions are applied.

8.3.7 Potential tuning left to be done

Even if a codec specification is frozen, encoders still have the flexibility to improve its implementation. In Linelet, it is up to the encoder to decide how sub-bands should be quantized in order to meet the optimal quality at a given bitrate.

This Linelet software implementation is very primitive in this regard. It follows a fixed pattern in how to add distortion to a slice in order to meet bitrate targets. It does not attempt to take into consideration energy distribution of sub-bands or any kind of adaptive method. It would likely be too slow anyways.

However, this ”blindness” of Linelet can be a strength in some sense. A common mistake of many codecs, DCT and wavelet alike, is to only optimize for PSNR in which the quantizer is lead to quantize in such a way that far too many high frequency components are zeroed out because large rate distortion gains can be made locally. The PSNR metric tends to ”like” blur, but human eyes do not. In flatter regions, these quantizers can easily turn textures into completely ”flat” and blurred out regions. This loss of visual energy can be very problematic for visual quality even if PSNR remains high.

Chapter 9

Conclusion

To address growing needs in the broadcasting industry, we designed, specified, implemented, optimized and evaluated a new lightweight compression codec, Linelet, which targets compression ratios 1:2 to 1:4 while remaining visually lossless. The codec was designed entirely with processing efficiency in mind. As a result, the implementation can encode or decode 1080p50 10-bit 4:2:2 and beyond in real-time on regular desktop PC hardware. There are strong indications that 4K@60 encoding is possible in real-time on powerful desktop equipment.

To evaluate our work, we designed a subjective evaluation based on ITU-R BT.500 with help from experts in the field. Two experts helped evaluate the Linelet implementation using two very critical test sequences, *ParkJoy* and *Horse*. The tests indicate that 1:2 and 1:4, and even 1:6 compression are visually lossless. More testing is required to ensure that this test result really holds in practice across a large range of video sequences and expert evaluations.

Linelet supports a simple method to exploit vertical redundancy within slices. We found that just by increasing slice height from 1 to 2 scanlines, we could obtain a 3 dB PSNR gain. Going beyond 2 scanlines further improves compression, but the gains beyond 2 scanlines are not as significant. We consider vertical transforms to be a useful coding tool for ultra-low latency video codecs.

To further improve vertical transforms, we implemented a pre/post filtering structure. We found there is some coding gain with pre/post filtering, but it depends on many factors. This filter works best at lower bitrates, far below the target 1:2 to 1:4 compression rates.

We tested multiple-generation loss with same coding parameters for all generations, and found that without pre/post filtering, Linelet is essentially mathematically lossless. With pre/post filtering added, a minimal 0.05 dB loss was observed over 7 generations.

There are still some Linelet codec design considerations to be made, but such decisions can be made during implementation in hardware. While Linelet was implemented in software here, the design was made to facilitate easy and efficient hardware implementation. We have outlined some of these concerns in the discussion chapter.

We propose that Linelet is a useful contribution in the area of low-latency video compression and is a candidate to be implemented in production. We believe the requirements for a lightweight production codec are fulfilled.

Bibliography

[1] http://upload.wikimedia.org/wikipedia/commons/2/23/Dctjpeg.png, [Online; accessed 2014-03-19].

[2] http://people.xiph.org/~xiphmont/demo/daala/screendoor.png, [Online; accessed 2014-03-19].

[3] http://en.wikipedia.org/wiki/File:Jpeg2000_2-level_wavelet_transform-lichtenstein.png, [Online; accessed 2014-03-24].

[4] http://en.wikipedia.org/wiki/File:STFT_and_WT.jpg, [Online; accessed 2014-03-19].

[5] J. Liang, C. Tu, and T. Tran, "Optimal block boundary pre/post-filtering for wavelet-based image and video compression," in *Image Processing, 2004. ICIP '04. 2004 International Conference on*, vol. 1, Oct 2004, pp. 303–306 Vol. 1, [Online version: http://thanglong.ece.jhu.edu/Tran/Pub/wtpre.pdf; accessed 2014-03-28].

[6] ITU-R, "BT.709-5: Parameter values for the HDTV standards for production and international programme exchange."

[7] ——, "BT.2020: Parameter values for ultra-high definition television systems for production and international programme exchange."

[8] http://www.nhk.or.jp/strl/english/aboutstrl1/r1.htm, [Online; accessed 2014-05-19].

[9] http://www.satellitemarkets.com/news-analysis/4k-tv-technology-push-or-demand-pull, [Online; accessed 2014-05-21].

[10] "3D TV is dead," http://www.extremetech.com/extreme/145168-3d-tv-is-dead, [Online; accessed 2014-04-25].

[11] ISO and ITU-T, "H.265: High efficiency video coding," 2013.

[12] ——, "MPEG-4 Part 10 (AVC) / H.264," 2003.

[13] http://www.webmproject.org/vp9/, [Online; accessed 2014-04-02].

[14] http://people.xiph.org/~xiphmont/demo/daala/demo1.shtml, [Online; accessed 2014-03-19].

[15] Peter Schelkens, Athanassios Skodras, and Touradj Ebrahimi, *The JPEG 2000 Suite*, 2009, pp. 393-395.

[16] ISO/IEC, "JPEG 2000 Core coding system (Part 1)," 2000.

[17] "Video Services Forum - About VSF]," http://www.videoservicesforum.org/about_vsf.shtml, [Online; accessed 2013-10-03].

[18] Video Services Forum, “Transport of JPEG 2000 Broadcast Profile video in MPEG-2 TS over IP,” http://www.videoservicesforum.org/activity_groups/VSF_TR-01_2013-04-15.pdf, Tech. Rep., 2013, [Online; accessed 2013-09-12].

[19] http://www.youtube.com/watch?v=HVFFq44UvLA, [Online; accessed 2014-05-26].

[20] http://www.intopix.com/uploaded/Download%20Products/intoPIX-TICO%20FLYER_ALTERA.pdf, [Online; accessed 2014-04-02].

[21] https://tech.ebu.ch/docs/techreview/trev_2012-Q4_SDI-over-IP_Laabs.pdf, [Online; accessed 2014-05-09].

[22] http://www.videoservicesforum.org/jt-nm/phase1.shtml, [Online; accessed 2014-05-09].

[23] “VESA Display Stream Compression,” http://www.vesa.org/featured-articles/vesa-and-mipi-alliance-announce-the-adoption-of-vesas-new-display-stream-compression-standard/, [Online; accessed 2014-04-26].

[24] http://www.hdmi.org/manufacturer/hdmi_2_0/, [Online; accessed 2014-05-23].

[25] “About Dirac,” http://diracvideo.org/about-dirac/, [Online; accessed 2014-04-02].

[26] ITU-R, “BT.500-13: Methodology for the subjective assessment of the quality of television pictures.”

[27] N. Ahmed, T. Natarajan, and K. Rao, “Discrete cosine transform,” *Computers, IEEE Transactions on*, vol. C-23, no. 1, pp. 90–93, Jan 1974.

[28] S. Mallat, “Multifrequency channel decompositions of images and wavelet models,” *Acoustics, Speech and Signal Processing, IEEE Transactions on*, vol. 37, no. 12, pp. 2091–2110, Dec 1989.

[29] Claude E. Shannon, *A Mathematical Theory of Communication*, 1948.

[30] David S. Taubman and Michael W. Marcellin, *JPEG2000 - Image compression fundamentals, standards and practice*, 2002, pp. 106-107.

[31] Peter Schelkens, Athanassios Skodras, and Touradj Ebrahimi, *The JPEG 2000 Suite*, 2009, pp. 390.

[32] O. Alay and H. Stephansen, “The effect of multi-generation encoding in broadcast contribution on the end-user video quality,” in *Packet Video Workshop (PV), 2012 19th International*, 2012, pp. 113–118.

[33] J. Shapiro, “An embedded hierarchical image coder using zerotrees of wavelet coefficients,” in *Data Compression Conference, 1993. DCC ’93.*, 1993, pp. 214–223.

[34] A. Said and W. Pearlman, “A new, fast, and efficient image codec based on set partitioning in hierarchical trees,” *Circuits and Systems for Video Technology, IEEE Transactions on*, vol. 6, no. 3, pp. 243–250, Jun 1996.

[35] “Dirac Specification,” http://diracvideo.org/download/specification/dirac-spec-latest.pdf, [Online; accessed 2014-04-10].

[36] “JTNM request for technology, intoPIX,” http://videoservicesforum.org/download/jtnm/JTNM012-1.zip, [Online; accessed 2014-04-25].

[37] ITU-R, “BT.601-7: Studio encoding parameters of digital television for standard 4:3 and wide screen 16:9 aspect ratios.”

[38] Jose Oliver and M. P. Malumbres, “A fast wavelet transform for image coding with low memory consumption,” 2004.

[39] Peter Schelkens, Athanassios Skodras, and Touradj Ebrahimi, *The JPEG 2000 Suite*, 2009, pp. 95-97.

[40] David S. Taubman and Michael W. Marcellin, *JPEG2000 - Image compression fundamentals, standards and practice*, 2002, pp. 193.

[41] “CUDA JPEG encoder,” http://www.fastvideo.ru/english/products/software/cuda-jpeg-encoder.htm, [Online; accessed 2014-03-31].

[42] M. Nadenau, J. Reichel, and M. Kunt, “Wavelet-based color image compression: exploiting the contrast sensitivity function,” *Image Processing, IEEE Transactions on*, vol. 12, no. 1, pp. 58–70, Jan 2003.

[43] J. Mannos and D. Sakrison, “The effects of a visual fidelity criterion of the encoding of images,” *Information Theory, IEEE Transactions on*, vol. 20, no. 4, pp. 525–536, Jul 1974.

[44] ITU-R, “BT.814-2: Specifications and alignment procedures for setting of brightness and contrast of displays.”

[45] “DCI Standard Evaluation Material,” http://www.dcimovies.com/StEM/, [Online; accessed 2014-03-05].

[46] ftp://vqeg.its.bldrdoc.gov/HDTV/SVT_MultiFormat/1080p50_CgrLevels_SINC_FILTER_SVTdec05_/2_ParkJoy_1080p50_CgrLevels_SINC_FILTER_SVTdec05_/, [Online; accessed 2014-03-30].

[47] Z. Wang, A. Bovik, H. Sheikh, and E. Simoncelli, “Image quality assessment: from error visibility to structural similarity,” *Image Processing, IEEE Transactions on*, vol. 13, no. 4, pp. 600–612, April 2004.

[48] Fitri N. Rahayu, Ulrich Reiter, Touradj Ebrahimi, Andrew Perkis, and Peter Svensson, “SS-SSIM and MS-SSIM for Digital Cinema Applications,” 2009.

Appendix A

Creating test sequences

A.1 Creating raw YUV source

```
# The sequence of images was concatenated to create the video stream.
ffmpeg -i input.tif -f rawvideo -c:v rawvideo \
   -vf scale=2048:857:out_color_matrix=bt709 -pix_fmt yuv422p10le raw_2k.yuv
# Crop to 1080p
ffmpeg -f rawvideo -s 2048x857 -pix_fmt yuv422p10le \
   -i raw_2k.yuv -vf crop=1920:816:64:25 raw_1080p.yuv
```

A.2 Creating mathematically lossless reference source

```
# colorspace 1 is BT.709 in FFmpeg.
ffmpeg -f rawvideo -r 24 -pix_fmt yuv422p10le -s 1920x816 -colorspace 1 \
   -i raw_1080p.yuv -c:v liblinelet -q:v 0 test_reference.llv
```

A.3 Creating lossy tests

```
# 100 Mbit/s.
ffmpeg -f rawvideo -r 24 -pix_fmt yuv422p10le -s 1920x816 -colorspace 1 \
   -i raw_1080p.yuv -c:v liblinelet -b:v 100000k \
   -slice_height 16 -precinct_samples 64 -slice_filter 3 test_100.llv
```

The linelet codec settings used were

- 16 scanlines per slice
- 64 samples precinct
- 5 horizontal decompositions
- 8 line pre/post filter

Appendix B

Evaluation schema

Name / date

Imperceptible	100
Perceptible, not annoying	80
Slightly annoying	60
Annoying	40
Very annoying	20
(Lowest possible score)	0

	Duration (s)
Reference sequence	**10**
Gray tone	3
Test sequence	**10**
Gray tone	3
Reference sequence	**10**
Gray tone	3
Test sequence	**10**
Gray tone / **Vote**	**8**

Test clip No.	**Score**		
1		21	
2		22	
3		23	
4		24	
5		25	
6		26	
7		27	
8		28	
9		29	
10		30	
11		31	
12		32	
13		33	
14		34	
15		35	
16		36	
17			
18			
19			
20			

Appendix C

Noise power normalization code

```matlab
function [even, odd] = lift_forward(sig)

even = sig(1 : 2 : end);
odd = sig(2 : 2 : end);

% Forward 5/3
odd = odd - 0.5 * even - 0.5 * [even(2 : end) even(end)];
even = even + 0.25 * [odd(1) odd(1 : end - 1)] + 0.25 * odd;
```

matlab/lift_forward.m

```matlab
function sig = lift_inverse(even, odd)

% Inverse 5/3
even = even - 0.25 * [odd(1) odd(1 : end - 1)] - 0.25 * odd;
odd = odd + 0.5 * even + 0.5 * [even(2 : end) even(end)];

sig = zeros(1, length(even) + length(odd));
sig(1 : 2 : end) = even;
sig(2 : 2 : end) = odd;
```

matlab/lift_inverse.m

```matlab
function [ll, lh, hl, hh] = lift_forward_2d(sig)

% Vertical forward lift
even = sig(1 : 2 : end, :);
odd = sig(2 : 2 : end, :);

odd = odd - 0.5 * even - 0.5 * [even(2 : end, :); even(end, :)];
even = even + 0.25 * [odd(1, :); odd(1 : end - 1, :)] + 0.25 * odd;

% Horizontal lifts
ll = even(:, 1 : 2 : end);
hl = even(:, 2 : 2 : end);
hl = hl - 0.5 * ll - 0.5 * [ll(:, 2 : end) ll(:, end)];
ll = ll + 0.25 * [hl(:, 1) hl(:, 1 : end - 1)] + 0.25 * hl;

lh = odd(:, 1 : 2 : end);
hh = odd(:, 2 : 2 : end);
hh = hh - 0.5 * lh - 0.5 * [lh(:, 2 : end) lh(:, end)];
lh = lh + 0.25 * [hh(:, 1) hh(:, 1 : end - 1)] + 0.25 * hh;
```

matlab/lift_forward_2d.m

```
function sig = lift_inverse_2d(ll, lh, hl, hh)

% Horizontal inverse lifts
ll = ll - 0.25 * [hl(:, 1) hl(:, 1 : end - 1)] - 0.25 * hl;
hl = hl + 0.5 * ll + 0.5 * [ll(:, 2 : end) ll(:, end)];

lh = lh - 0.25 * [hh(:, 1) hh(:, 1 : end - 1)] - 0.25 * hh;
hh = hh + 0.5 * lh + 0.5 * [lh(:, 2 : end) lh(:, end)];

even = zeros(size(ll, 1), size(ll, 2) + size(hl, 2));
odd = zeros(size(lh, 1), size(lh, 2) + size(hh, 2));

even(:, 1 : 2 : end) = ll;
even(:, 2 : 2 : end) = hl;
odd(:, 1 : 2 : end) = lh;
odd(:, 2 : 2 : end) = hh;

% Vertical inverse lifts
even = even - 0.25 * [odd(1, :); odd(1 : end - 1, :)] - 0.25 * odd;
odd = odd + 0.5 * even + 0.5 * [even(2 : end, :); even(end, :)];


sig = zeros(size(even, 1) + size(odd, 1), size(even, 2));
sig(1 : 2 : end, :) = even;
sig(2 : 2 : end, :) = odd;
```

matlab/lift_inverse_2d.m

```
power_spectrum = zeros(1, 8 * 1024);
window = kaiser(8 * 1024, 20.0)';

for i = 1 : 1024
    sig = randn(1, 8 * 1024);

    [l1, h1] = lift_forward(sig);
    [l2, h2] = lift_forward(l1);
    [l3, h3] = lift_forward(l2);
    [l4, h4] = lift_forward(l3);
    [l5, h5] = lift_forward(l4);

    q = 0.005;

    h1 = h1 + q * (rand(1, length(h1)) - 0.5);
    h2 = h2 + q * (rand(1, length(h2)) - 0.5);
    h3 = h3 + q * (rand(1, length(h3)) - 0.5);
    h4 = h4 + q * (rand(1, length(h4)) - 0.5);
    h5 = h5 + q * (rand(1, length(h5)) - 0.5);
    l5 = l5 + q * (rand(1, length(l5)) - 0.5);

    l4 = lift_inverse(l5, h5);
    l3 = lift_inverse(l4, h4);
    l2 = lift_inverse(l3, h3);
    l1 = lift_inverse(l2, h2);
    sig_out = lift_inverse(l1, h1);

    power_spec = fft((sig_out - sig) .* window);
    power_spec = power_spec .* conj(power_spec);
    power_spectrum = power_spec + power_spectrum;
end
```

```
power_spectrum = power_spectrum(1 : end / 2);
plot(linspace(0, 1, length(power_spectrum)), 10 * log10(power_spectrum));
xlabel('Normalized frequency (angular / pi)');
ylabel('Gain (dB)');
title('Synthesis filter gain Le Gall 5/3 (5 levels)');
saveas(gcf, 'legall_noise.eps');

power_spectrum = zeros(1, 8 * 1024);
window = kaiser(8 * 1024, 20.0)';

for i = 1 : 1024
    sig = randn(1, 8 * 1024);

    [l1, h1] = lift_forward(sig);
    [l2, h2] = lift_forward(l1);
    [l3, h3] = lift_forward(l2);
    [l4, h4] = lift_forward(l3);
    [l5, h5] = lift_forward(l4);

    q = 0.005;

    % Add error (weighed)
    h1 = h1 + 32 * q * (rand(1, length(h1)) - 0.5);
    h2 = h2 + 16 * q * (rand(1, length(h2)) - 0.5);
    h3 = h3 +  8 * q * (rand(1, length(h3)) - 0.5);
    h4 = h4 +  4 * q * (rand(1, length(h4)) - 0.5);
    h5 = h5 +  2 * q * (rand(1, length(h5)) - 0.5);
    l5 = l5 +  1 * q * (rand(1, length(l5)) - 0.5);

    l4 = lift_inverse(l5, h5);
    l3 = lift_inverse(l4, h4);
    l2 = lift_inverse(l3, h3);
    l1 = lift_inverse(l2, h2);
    sig_out = lift_inverse(l1, h1);

    power_spec = fft((sig_out - sig) .* window);
    power_spec = power_spec .* conj(power_spec);
    power_spectrum = power_spec + power_spectrum;
end

power_spectrum = power_spectrum(1 : end / 2);
plot(linspace(0, 1, length(power_spectrum)), 10 * log10(power_spectrum));
xlabel('Normalized frequency (angular / pi)');
ylabel('Gain (dB)');
title('Synthesis filter gain Le Gall 5/3 (5 levels, 1 bit per level)');
saveas(gcf, 'legall_noise_weighed.eps');

power_spectrum = zeros(1, 8 * 1024);
window = kaiser(8 * 1024, 20.0)';

for i = 1 : 1024
    sig = randn(1, 8 * 1024);

    [l1, h1] = lift_forward(sig);
    [l2, h2] = lift_forward(l1);
    [l3, h3] = lift_forward(l2);
    [l4, h4] = lift_forward(l3);
    [l5, h5] = lift_forward(l4);

    q = 0.005 * sqrt(2);

    % Add error (weighed)
    h1 = h1 +         1 * 32 * q * (rand(1, length(h1)) - 0.5);
    h2 = h2 +   sqrt(2) * 16 * q * (rand(1, length(h2)) - 0.5);
    h3 = h3 +   sqrt(4) *  8 * q * (rand(1, length(h3)) - 0.5);
    h4 = h4 +   sqrt(8) *  4 * q * (rand(1, length(h4)) - 0.5);
    h5 = h5 + sqrt(16) *  2 * q * (rand(1, length(h5)) - 0.5);
    l5 = l5 + sqrt(16) *  1 * q * (rand(1, length(l5)) - 0.5);

    l4 = lift_inverse(l5, h5);
    l3 = lift_inverse(l4, h4);
    l2 = lift_inverse(l3, h3);
    l1 = lift_inverse(l2, h2);
    sig_out = lift_inverse(l1, h1);

    power_spec = fft((sig_out - sig) .* window);
    power_spec = power_spec .* conj(power_spec);
    power_spectrum = power_spec + power_spectrum;
end

power_spectrum = power_spectrum(1 : end / 2);
plot(linspace(0, 1, length(power_spectrum)), 10 * log10(power_spectrum));
xlabel('Normalized frequency (angular / pi)');
ylabel('Gain (dB)');
title('Synthesis filter gain Le Gall 5/3 (5 levels, fully weighed)');
saveas(gcf, 'legall_noise_weighed_size.eps');
```

matlab/noise_power.m

```
%%
points = 64;
power_spectrum = zeros(points, points);
window = kaiser(points, 5.0) * kaiser(points, 5.0)';

for i = 1 : 1024
    sig = randn(points, points);

    [ll, lh, hl, hh] = lift_forward_2d(sig);

    q = 0.005;
    ll = ll + q * (rand(size(ll)) - 0.5);
    lh = lh + q * (rand(size(lh)) - 0.5);
    hl = hl + q * (rand(size(hl)) - 0.5);
    hh = hh + q * (rand(size(hh)) - 0.5);

    sig_out = lift_inverse_2d(ll, lh, hl, hh);

    power_spec = fft2((sig_out - sig) .* window);
    power_spec = power_spec .* conj(power_spec);
    power_spectrum = power_spec + power_spectrum;
end

power_spectrum = fftshift(power_spectrum);

%%

[x, y] = meshgrid(-points / 2 : points / 2 - 1);
x = x / (points / 2);
y = y / (points / 2);

power = 10 * log10(power_spectrum);
```

```
surf(x, y, power);
colorbar;
xlabel('Horizontal freq. (angular / pi)');
ylabel('Vertical freq. (angular / pi)');
zlabel('Gain (dB)');
title('Synthesis filter gain 2D Le Gall 5/3');
%saveas(gcf, 'legall_noise_2d.eps');

%%
points = 64;
power_spectrum = zeros(points, points);
window = kaiser(points, 5.0) * kaiser(points, 5.0)';

for i = 1 : 1024
    sig = randn(points, points);

    [ll, lh, hl, hh] = lift_forward_2d(sig);

    q = 0.005;
    ll = ll + 1 * q * (rand(size(ll)) - 0.5);
    lh = lh + 2 * q * (rand(size(lh)) - 0.5);
    hl = hl + 2 * q * (rand(size(hl)) - 0.5);
    hh = hh + 4 * q * (rand(size(hh)) - 0.5);

    sig_out = lift_inverse_2d(ll, lh, hl, hh);

    power_spec = fft2((sig_out - sig) .* window);
    power_spec = power_spec .* conj(power_spec);
    power_spectrum = power_spec + power_spectrum;
end

power_spectrum = fftshift(power_spectrum);

%%

[x, y] = meshgrid(-points / 2 : points / 2 - 1);
x = x / (points / 2);
y = y / (points / 2);

power = 10 * log10(power_spectrum);

surf(x, y, power);
colorbar;
xlabel('Horizontal freq. (angular / pi)');
ylabel('Vertical freq. (angular / pi)');
zlabel('Gain (dB)');
title('Synthesis filter gain 2D Le Gall 5/3, weighed');
%saveas(gcf, 'legall_noise_2d_weighed.eps');

%%
points = 128;
power_spectrum = zeros(points, points);
window = kaiser(points, 5.0) * kaiser(points, 5.0)';

for i = 1 : 1024
    sig = randn(points, points);

    [ll1, lh1, hl1, hh1] = lift_forward_2d(sig);
    [ll2, lh2, hl2, hh2] = lift_forward_2d(ll1);
    [ll3, lh3, hl3, hh3] = lift_forward_2d(ll2);
    [ll4, lh4, hl4, hh4] = lift_forward_2d(ll3);
    [ll5, lh5, hl5, hh5] = lift_forward_2d(ll4);

    q = 0.005;
    ll5 = ll5 + 32 * 1 * q * (rand(size(ll5)) - 0.5);

    lh5 = lh5 + 32 * 2 * q * (rand(size(lh5)) - 0.5);
    hl5 = hl5 + 32 * 2 * q * (rand(size(hl5)) - 0.5);
    hh5 = hh5 + 32 * 4 * q * (rand(size(hh5)) - 0.5);

    lh4 = lh4 + 16 * 8 * q * (rand(size(lh4)) - 0.5);
    hl4 = hl4 + 16 * 8 * q * (rand(size(hl4)) - 0.5);
    hh4 = hh4 + 16 * 16 * q * (rand(size(hh4)) - 0.5);

    lh3 = lh3 + 8 * 32 * q * (rand(size(lh3)) - 0.5);
    hl3 = hl3 + 8 * 32 * q * (rand(size(hl3)) - 0.5);
    hh3 = hh3 + 8 * 64 * q * (rand(size(hh3)) - 0.5);

    lh2 = lh2 + 4 * 128 * q * (rand(size(lh2)) - 0.5);
    hl2 = hl2 + 4 * 128 * q * (rand(size(hl2)) - 0.5);
    hh2 = hh2 + 4 * 256 * q * (rand(size(hh2)) - 0.5);

    lh1 = lh1 + 2 * 512 * q * (rand(size(lh1)) - 0.5);
    hl1 = hl1 + 2 * 512 * q * (rand(size(hl1)) - 0.5);
    hh1 = hh1 + 2 * 1024 * q * (rand(size(hh1)) - 0.5);

    ll4 = lift_inverse_2d(ll5, lh5, hl5, hh5);
    ll3 = lift_inverse_2d(ll4, lh4, hl4, hh4);
    ll2 = lift_inverse_2d(ll3, lh3, hl3, hh3);
    ll1 = lift_inverse_2d(ll2, lh2, hl2, hh2);
    sig_out = lift_inverse_2d(ll1, lh1, hl1, hh1);

    power_spec = fft2((sig_out - sig) .* window);
    power_spec = power_spec .* conj(power_spec);
    power_spectrum = power_spec + power_spectrum;
end

power_spectrum = fftshift(power_spectrum);

%%

[x, y] = meshgrid(-points / 2 : points / 2 - 1);
x = x / (points / 2);
y = y / (points / 2);

power = 10 * log10(power_spectrum);

surf(x, y, power);
colorbar;
xlabel('Horizontal freq. (angular / pi)');
ylabel('Vertical freq. (angular / pi)');
zlabel('Gain (dB)');
title('Synthesis filter gain 2D Le Gall 5/3, (5 levels, weighed)');
%saveas(gcf, 'legall_noise_2d_weighed_size.eps');
```

matlab/noise_power_2d.m

```
%% Unbiased

lo_pass = fft([1/2 1 1/2], 256);
lo_pass = lo_pass .* conj(lo_pass);
hi_pass = fft([-1/8 -2/8 6/8 -2/8 -1/8], 256);
hi_pass = hi_pass .* conj(hi_pass);

spectrum_est = lo_pass + hi_pass;
spectrum_est = spectrum_est(1 : end / 2);

power_spectrum = power_spectrum(1 : end / 2);

figure;
plot(linspace(0, 1, length(spectrum_est)), 10 * log10(spectrum_est), 'k');
xlabel('Normalized frequency (angular / pi)');
ylabel('Gain (dB)');
title('Synthesis filter gain Le Gall 5/3');

saveas(gcf, 'legall_basic.eps');

%% One-bit high-pass bias
lo_pass = fft([1/2 1 1/2], 256);
lo_pass = lo_pass .* conj(lo_pass);
hi_pass = fft(2 * [-1/8 -2/8 6/8 -2/8 -1/8], 256);
hi_pass = hi_pass .* conj(hi_pass);

spectrum_est = lo_pass + hi_pass;
spectrum_est = spectrum_est(1 : end / 2);

power_spectrum = power_spectrum(1 : end / 2);

figure;
plot(linspace(0, 1, length(spectrum_est)), 10 * log10(spectrum_est), 'k');
xlabel('Normalized frequency (angular / pi)');
ylabel('Gain (dB)');
title('Synthesis filter gain Le Gall 5/3 (1 bit extra quantization on high-pass)
    ');
saveas(gcf, 'legall_basic_bias.eps');
```

matlab/noise_power_basic.m

```
f = linspace(0, 40, 1024);

csf = 2.6 * (0.0192 + 0.114 * f) .* exp(-((0.114 * f) .^ 1.1));
b = log2(1 ./ csf);
%b = b - min(b);

plot(f, b);
xlabel('Cycles per degree (cpd)');
ylabel('Bits');
title('Luminance added quantization bits');
saveas(gcf, 'csf-quant.eps');

cpdmax = 49;
freqs = 1/64 * [
     1   1
     3   1
     1   3
     3   3
     6   2
     2   6
     6   6
    12   4
     4  12
    12  12
    24   8
     8  24
    24  24
    48  16
    16  48
    48  48]';

freqs_out = sqrt(sum(freqs .* freqs, 1));
```

matlab/csf.m

Appendix D

Subjective and objective evaluation test data

```
rates = [2.5  5.0  10.0  15.0  20.0  25.0  35.0  50.0  100.0];

dci1 = [
    10  20
    60  50
    80  80
    90  80
    90  90
    90  82
    90  85
    90  80
    90  80
];

dci2 = [
    40  85
    70  90
    90  100
    85  90
    90  90
    90  80
    90  100
    90  90
    80  80
];

parkjoy = [
    5  10
    50  40
    75  70
    90  90
    90  100
    90  82
    90  85
    90  100
    90  85
];

horse = [
    0  20
    10  30
    40  50
    80  85
    90  90
    80  82
    85  100
    90  82
    80  90
];

psnr = [
    23.467  26.662  30.109  33.755  37.131  40.287  46.626  55.705 % ParkJoy
    25.460  28.026  31.466  34.794  38.504  42.588  49.098  58.324 % Horse
    34.484  38.777  42.631  45.532  48.376  51.262  57.085  65.571 % DCI1
    38.068  41.160  44.037  46.812  49.110  52.317  57.795  68.198 % DCI2
];

ssim = [
     5.40   8.23  11.19  13.92  16.52  19.10  24.40  32.95 % ParkJoy
     5.81   8.04  10.60  13.00  15.67  19.40  25.39  34.63 % Horse
    11.54  14.52  17.15  19.41  21.89  24.66  30.09  38.84 % DCI1
    11.11  13.64  16.12  18.77  20.94  24.06  29.46  40.16 % DCI2
];

%% Horse
figure;
hold on;
plot(rates(1 : end), dci2(:, 1)' / 20, 'k-^');
plot(rates(1 : end), dci2(:, 2)' / 20, 'k-s');
hold off;
grid on;

xlabel('Compression rate (%)');
ylabel('Opinion score');

legend('Expert 1', 'Expert 2', 'Location', 'SouthEast');
axes_pos = get(gca, 'Position');
top_x = axes('Position', axes_pos, 'Color', 'none', 'XLim', [0 20], '
    XAxisLocation', 'top', 'YAxisLocation', 'right', 'YTick', []);
h = xlabel(top_x, 'Bits per pixel');
P = get(h, 'Position');
set(h, 'Position', [P(1), P(2) - 0.04, P(3)]);

%% PSNR
figure;
hold on;
plot(rates(1 : end - 1), psnr(1, :), 'k-+');
plot(rates(1 : end - 1), psnr(2, :), 'k-d');
plot(rates(1 : end - 1), psnr(3, :), 'k-s');
plot(rates(1 : end - 1), psnr(4, :), 'k-^');
hold off;
grid on;
xlabel('Compression rate (%)');
ylabel('PSNR-Y (dB)');

h = legend('ParkJoy', 'Horse', 'DCI1', 'DCI2');
set(h, 'Location', 'southeast');
%h = title('PSNR-Y for test sequences at tested bitrates');
axes_pos = get(gca, 'Position');
top_x = axes('Position', axes_pos, 'Color', 'none', 'XLim', [0 10], '
    XAxisLocation', 'top', 'YAxisLocation', 'right', 'YTick', []);
h = xlabel(top_x, 'Bits per pixel');
P = get(h, 'Position');
set(h, 'Position', [P(1), P(2) - 0.02, P(3)]);
```

```
saveas(gcf, 'psnr.eps');

%% SSIM
figure;
hold on;
plot(rates(1 : end - 1), ssim(1, :), 'k-+');
plot(rates(1 : end - 1), ssim(2, :), 'k-d');
plot(rates(1 : end - 1), ssim(3, :), 'k-s');
plot(rates(1 : end - 1), ssim(4, :), 'k-^');
hold off;
grid on;
xlabel('Compression rate (%)');
ylabel('SSIM (dB) [-10log10(1 - SSIM)]');

h = legend('ParkJoy', 'Horse', 'DCI1', 'DCI2');
set(h, 'Location', 'southeast');
%title('SS-SSIM for test sequences at tested bitrates');
axes_pos = get(gca, 'Position');
top_x = axes('Position', axes_pos, 'Color', 'none', 'XLim', [0 10], '
    XAxisLocation', 'top', 'YAxisLocation', 'right', 'YTick', []);
h = xlabel(top_x, 'Bits per pixel');
P = get(h, 'Position');
set(h, 'Position', [P(1), P(2) - 0.02, P(3)]);

saveas(gcf, 'ssim.eps');

%% All
scores = [dci1'; dci2'; parkjoy'; horse'] / 20;
scores_mean = mean(scores, 1);
scores_dev = 1.96 * std(scores, 0, 1) / sqrt(size(scores, 1));

scores_mean_overall = mean(scores_mean);

figure;
hold on;
plot(rates, scores_mean - scores_dev, 'k-+');
plot(rates, scores_mean,              'k-d');
plot(rates, scores_mean + scores_dev, 'k-s');
hold off;
grid on;
xlabel('Compression rate (%)');
ylabel('Mean opinion score');
h = legend('Lower bound 95% confidence (mean)', 'Mean', 'Upper bound 95%
    confidence (mean)');
set(h, 'Location', 'southeast');
%title('Mean opinion scores over all sequences and observers');

axes_pos = get(gca, 'Position');
top_x = axes('Position', axes_pos, 'Color', 'none', 'XLim', [0 20], '
    XAxisLocation', 'top', 'YAxisLocation', 'right', 'YTick', []);
h = xlabel(top_x, 'Bits per pixel');
P = get(h, 'Position');
set(h, 'Position', [P(1), P(2) - 0.02, P(3)]);

saveas(gcf, 'all.eps');

%% DCI only
scores = [dci1'; dci2'] / 20;
scores_mean = mean(scores, 1);
scores_dev = 1.96 * std(scores, 0, 1) / sqrt(size(scores, 1));
```

```
figure;
hold on;
plot(rates, scores_mean - scores_dev, 'k-+');
plot(rates, scores_mean,              'k-d');
plot(rates, scores_mean + scores_dev, 'k-s');
hold off;
grid on;
xlabel('Compression rate (%)');
ylabel('Mean opinion score');
h = legend('Lower bound 95% confidence (mean)', 'Mean', 'Upper bound 95%
    confidence (mean)');
set(h, 'Location', 'southeast');
%title('Mean opinion scores over DCI sequences and observers');

axes_pos = get(gca, 'Position');
top_x = axes('Position', axes_pos, 'Color', 'none', 'XLim', [0 20], '
    XAxisLocation', 'top', 'YAxisLocation', 'right', 'YTick', []);
h = xlabel(top_x, 'Bits per pixel');
P = get(h, 'Position');
set(h, 'Position', [P(1), P(2) - 0.02, P(3)]);

saveas(gcf, 'DCI.eps');

%% SVT/NRK
scores = [parkjoy'; horse'] / 20;
scores_mean = mean(scores, 1);
scores_dev = 1.96 * std(scores, 0, 1) / sqrt(size(scores, 1));

figure;
hold on;
plot(rates, scores_mean - scores_dev, 'k-+');
plot(rates, scores_mean,              'k-d');
plot(rates, scores_mean + scores_dev, 'k-s');
hold off;
grid on;
xlabel('Compression rate (%)');
ylabel('Mean opinion score');
h = legend('Lower bound 95% confidence (mean)', 'Mean', 'Upper bound 95%
    confidence (mean)');
set(h, 'Location', 'southeast');
%title('Mean opinion scores over ParkJoy/Horse sequences and observers');

axes_pos = get(gca, 'Position');
top_x = axes('Position', axes_pos, 'Color', 'none', 'XLim', [0 20], '
    XAxisLocation', 'top', 'YAxisLocation', 'right', 'YTick', []);
h = xlabel(top_x, 'Bits per pixel');
P = get(h, 'Position');
set(h, 'Position', [P(1), P(2) - 0.02, P(3)]);

saveas(gcf, 'parkjoy_horse.eps');
```

subjective/subjective.m

```
psnr_500_0 = [39.674 39.674 39.674 39.674 39.674 39.674 39.674];
psnr_300_0 = [33.070 33.070 33.070 33.070 33.070 33.070 33.070];

psnr_500_8 = [39.619 39.606 39.594 39.583 39.574 39.567 39.559];
psnr_300_8 = [33.073 33.066 33.058 33.050 33.041 33.031 33.020];

generations = 1 : 7;
```

```
%% 500
figure;
hold on;
plot(generations, psnr_500_0, 'k-^');
plot(generations, psnr_500_8, 'k-s');
hold off;
legend('500 Mbit/s, no pre/post filter', '500 Mbit/s, 8-line pre/post filter');
xlabel('Generation');
ylabel('PSNR-Y (dB)');
title('Encoding loss over multiple generations for 500 Mbit/s ParkJoy');

saveas(gcf, 'parkjoy_500.eps');

%% 300

figure;
hold on;
plot(generations, psnr_300_0, 'k-^');
plot(generations, psnr_300_8, 'k-s');
hold off;
legend('300 Mbit/s, no pre/post filter', '300 Mbit/s, 8-line pre/post filter');
xlabel('Generation');
ylabel('PSNR-Y (dB)');
title('Encoding loss over multiple generations for 300 Mbit/s ParkJoy');

saveas(gcf, 'parkjoy_300.eps');
```

objective/objective.m

Appendix E

README.txt

```
=============================================
Additional material for Linelet master thesis
=============================================

In this archive, we present additional material which does fit
inside the report.

 - Linelet source code.
 - Linelet binaries for 64-bit Windows (Windows Vista or later).
 - Scripts which were used to run the subjective evaluation
   as well as objective metrics.

Test clips in RAW YUV format are *NOT* included here due to a
prohibitively large size. Which test clips were used is
laid out in the chapter 5 and can be obtained elsewhere.


Source code
================

Source code is found under linelet-source/linelet-source.zip.
The sole author of this code is Hans-Kristian Arntzen.
The exception to this is the xxhash implementation which is
licensed under BSD. A license header is included for this
in the relevant code.

In the source code, a patch for FFmpeg/libavcodec is also
included which is needed to build FFmpeg with Linelet support.
See the README.html included in the Linelet source.


Binaries
============

We include binaries for 64-bit Windows which can run on
Windows Vista or later. They are found in the binaries/ folder.
```

```
- linelet: The command-line utility for the Linelet codec,
  built from linelet/linelet-source.zip.

- FFmpeg: The command-line utility for FFmpeg.
  It was patched with Linelet support
  (linelet/linelet-source.zip)
  and otherwise built as-is with --enable-liblinelet.

- MPV: The MPV player. It was built as-is against dynamic
  FFmpeg libraries.

- Various open source dynamic libraries which are needed to
  run FFmpeg and MPV. They were built as-is.

Scripts
============

Scripts which were used to generate results for subjective
and objective evaluations (chapter 5 and 6) are found
in the scripts/ folder.
```