Guided Transcoding for Next-Generation Video Coding (HEVC)

Harald Nordgren
ada09hno@student.lth.se

March 13, 2016

Master's thesis work carried out at Ericsson AB

Supervisors:
Kenneth Andersson, kenneth.r.andersson@ericsson.com
Ruoyang Yu, ruoyang.yu@ericsson.com
Michael Doggett, michael.doggett@cs.lth.se

Examiner:
Jörn W. Janneck, jwj@cs.lth.se

Abstract

Video content is the dominant traffic type on mobile networks today, and that portion is only expected to increase in the future. In this thesis we investigate ways of reducing bit rates in adaptive streaming applications in the latest video coding standard, H.265 / High Efficiency Video Coding (HEVC).

The current models for dynamically offering different-resolution versions of video content, so called *adaptive streaming*, require either large amounts of storage capacity where full encodings of the material is kept at all times, or extremely high computational power in order to dynamically regenerate content on-demand.

Guided transcoding aims at finding a middle-ground were we can store and transmit less data, at full or near-full quality, while still keeping computational complexity low. This is achieved by shifting the computationally heavy operations to a preprocessing step where so called *side-information* is generated. The side-information can then be used to quickly reconstruct sequences on-demand – even when running on generic, non-specialized, hardware.

Two method for generating side-information, pruning and deflation, were compared on a varying set of standardized HEVC test sequences, and the respective upsides and downsides of each method are discussed.

Keywords: Video Compression, Transcoding, Adaptive Streaming, H.265, HEVC

Acknowledgements

I want to thank Ruoyang Yu for his continued support in writing code. I want to thank Kenneth Andersson for his help with the report and theory surrounding video coding. I want to thank Thomas Rusert, Markus Flierl. I want to thank Michael Dogget and Jörn Janneck at LTH for support with the thesis writing.

Contents

Acronyms

ABR Adaptive Bit Rate. 16
AVC Video coding standard (synonym of **H.264**). 15

CABAC Context-Adaptive Binary Arithmetic Coding. 11
CTB Coding Tree Block. 4, 5
CTU Coding Tree Unit.
CU Coding Unit.

DCT Discrete Cosine Transform. 9

fps Frames per Second. 3, 4, 25, 26

GOP Group of Pictures. 13, 23, 26
GT Guided Transcoding. 18, 19, 21, 22, 25, 30

H.264 Video coding standard. 11, 18
H.265 Video coding standard, successor of **H.264**. 18
HD High-Definition. 14, 15
HEVC Video coding standard (synonym of **H.265**). 4, 9, 11, 14, 15, 24–26
HM HEVC Test Model. 26
HQ High Quality. 22–24, 26–29

JIT Just-in-Time. 17–19, 21, 22
JVT Joint Video Team. 15, 25, 26

kbit/s Kilobits per Second. 29

LQ Low Quality. 19, 22–24, 26, 27, 29, 30

MSE Mean Squared Error. 7, 28

PSNR Peak Signal-to-Noise Ratio. 25, 27–30

QP Quantization Parameter. 9, 14, 19, 22, 25–28, 39

RDO Rate-Distortion Optimization. 9, 12, 15, 16
RDOQ Rate-Distortion Optimized Quantization. 15, 25, 30, 39
RGB Red, Green and Blue. 3, 4

SBH Sign-Bit Hiding. 16, 19, 25, 30
SDH Sign-Data Hiding (synonym of **SBH**).
SHVC Scalable High Efficiency Video Coding. 18
SI Side-Information. 21–23, 25–27, 29, 30
SVC Scalable Video Coding. 17–19, 21

YCbCr Color format often used instead of **RGB**. 3, 4

Chapter 1
Introduction

1.1 Video Codecs

Video is the dominant traffic type on mobile networks today and is expected to take up 70% of all traffic in a few years. In order to be able to store and transmit high-quality video the data needs to be encoded. Modern video coding algorithms can decrease the size of raw video by hundreds of times with little or no discernible loss in video quality.

To achieve this, local redundancies within each frame and between neighboring pictures are utilized, and different statistical methods guide the process to achieve the lowest file size at the least distortion of picture quality.

The standard scenario for digital video is for it to be captured by a camera, encoded and stored somewhere, and then transmitted over a network. On the receiving end the process is reversed, the sequences is decoded and then displayed.

An encoder always goes together with a decoder. Together the two are known as a *codec*. [3]

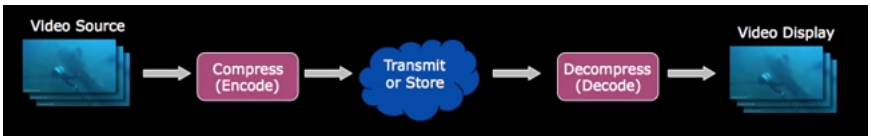

Figure 1.1: The basic idea of a codec

1.2 Images and Video Fundamentals

1.2.1 The Binary Number System

To store and transmit data in any computer system, numbers are converted to *binary format*. Instead of ten digits, 0–9, binary deals exclusively with two numbers: 0 and 1. [7] These can be more easily represented digitally.

A regular (decimal) representation of any number, for example the 312, gives significance to digits based on their positions. This is called a positional system. 312 can be re-written as

$$3 \times 100 + 1 \times 10 + 2 \times 1$$

which furthermore can be written as

$$3 \times 10^2 + 2 \times 10^1 + 1 \times 10^0$$

The internal logic of numbers is so ingrained into our everyday thinking that most never need to worry about it. Binary conversion, however, forces us examine the meaning of a positional system. Because binary only has two digits, each positional exponent is now a 2 instead of a 10. 312 would in binary be represented as

$$1 \times 2^8 + 0 \times 2^7 + 0 \times 2^6 + 1 \times 2^5 + 1 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 0 \times 2^0$$

which is equal to

$$1 \times 256 + 0 \times 128 + 0 \times 64 + 1 \times 32 + 1 \times 16 + 1 \times 8 + 0 \times 4 + 0 \times 2 + 0 \times 1$$

and which we write as as 100111000_2. The subscript is usually left out when it is clear from context that we are dealing with binary numbers.

The conversion itself is non-trivial, and we won't go into detail on how to actually convert 312_{10} to 100111000_2. It can easily be verified that the above sum adds up to the expected value, and it can be proven mathematically that each binary representation is unique.

1.2.2 Digitally Representing a Frame

The simplest form of digital image, a greyscale one, can be represented as a *light intensity function*. [10] For every x- and y-coordinate there is a light components representing brightness. The light intensity function exists in continuous space so there is though to be infinite precision.

This continuous-space representation is *sampled* at certain intervals to give us a digital representation. The samples are confined to some interval, usually a rectangular area that makes up the borders of the image.

Light samples are called pixels or pels, from the word *picture element*. The number of samples in a frame are usually written as $N \times M$, giving the number of horizontal versus vertical samples. [1] For example, 1920×1080. Often, the size of a frame is referred to only by its height.

1.2.3 Digitally Representing Video

What set video apart from images is movement. Images shown in rapid succession creates the illusion of movement as long as the *frame rate* is at least 24 frames per second (fps). [10] Higher frame rates give a smoother-looking video, so many systems use frame rates around 50-60 fps. HD or Ultra HD content will often have frame rates ranging from 100 to 200 fps.

1.2.4 RGB and YCbCr

The light intensity function represents a greyscale image. To achieve color, we sample many components simultaneously at different light frequencies and display them together.

Classically, three color samples, red, green and blue (RGB), are used. Strangely enough, this suffices to represent any color on the human visual spectrum. [1]

In modern applications, RGB is generally transformed to the YCbCr format. Y is the *luma* component, representing the overall brightness of the image, and Cb and Cr are *chroma* components, representing the intensity in blue and red. [10]

1.2.5 Chroma Subsampling

Because the human eye is more sensitive to light than color, [3][10] in YCbCr the two color components are usually *subsampled* so that chroma values are averaged from a neighborhood of pixels, and subsequently less information is transmitted for the chroma than for luma. [1]

This subsampling is conventionally written as a ratio between luma and chroma samples

$$Y : X_1 : X_2$$

with $Y = 4$ in most applications. X_1 describes horizontal subsampling, so $X_1 = 2$ means that 2 chroma samples are taken horizontally for every 4 luma samples. X_2 describes the vertical subsampling in relation to X_1.

$X_1 = 0$ indicates that the same chroma subsampling is performed vertically as horizontally, while $X_1 = 2$ indicates no vertical subsampling is performed – that is, the number of chroma samples are the same as the luma samples.

$4 : 2 : 2$ and $4 : 2 : 0$ are the most common subsampling formats. [10] In $4 : 2 : 0$, for example, for every $2 \times 2 = 4$ luma pixels, only 1 pixel is stored in the two chroma components.

This means that a 1920×1080 video subsampled at $4 : 2 : 2$ contains only 960×540 chroma samples. Naturally, this decreases the size of the stored video. Two chroma components each a one fourth's size plus the luma gives 1.5 full samples, as opposed to the 3 components needed in RGB.

Done correctly, chroma subsampling has little to no effect on the resultant video. Instead, for the same file size as without subsampling, the overall resolution and quality can now be increased.

But, even with RGB transformed to YCbCr, storing or transmitting each pixel value is expensive in terms of storage space and bandwidth. 10 seconds of $4 : 2 : 0$ video content

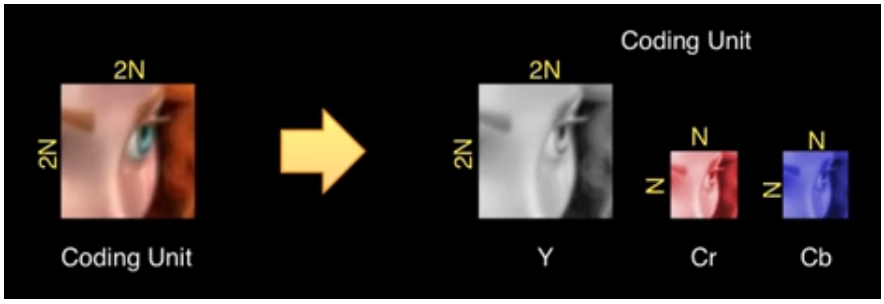

Figure 1.2: Dividing a coding unit into three components and subsampling

with 8 bit intensity values at 1920×1080 resolution at 50 fps will require $1.5 \times 10 \times 8 \times 1920 \times 1080 \times 50$ bits = 1.6 GB of data. A two hour movie would require 1.1 TB. This is why we need compression.

1.3 Encoding Fundamentals

A video encoder is a program that takes some form of raw video material and produces a *bitstream*, a sequence of 0's and 1's – commonly referred to as bits. The encoder does this by going over the video frame-by-frame, often in a non-sequential order, using a multitude of algorithms to codify and store the information using as few bits as possible.

A certain portion of encoding is *lossless*, that is, information is repackaged in a form so that it can be completely restored by the decoder. The majority of gains, however, are achieved through *lossy* compression, where data is irreparably distorted. A good coding algorithm achieves large decreases in file size without sacrificing too much quality. Any decrease in bitstream size is always weighed against quality loss it introduces.

A decoder turns the bitstream into pixel data again. High Efficiency Video Coding (HEVC), like most video coding standards, only specifies the behavior of its decoder. That is, a bitstream is said to conform to the HEVC standard as long as it can be decoded by its decoder, regardless of relative quality of the video output. The implementer of an encoder is free to make choices in the coding as long as the resulting bitstream conforms to the standard – which naturally limits the freedom.

1.3.1 Partitioning

The first step of the encoding process is partitioning. The HEVC encoder divides the frame into slices that can be encoded in parallel. This division is usually done once for the whole sequence.

Then, each slice goes through *quadtree* partionining. The slice is divided into 64×64 coding tree blocks (CTBs) [2] which are then recursively split into four equally sized subsections.

Both options – splitting and not splitting – are performed and encoded. The encoder compares the results and writes the most effective structure to the bitstream. For each 64×64 block that is split up, the procedure is repeated for each of the four subsections,

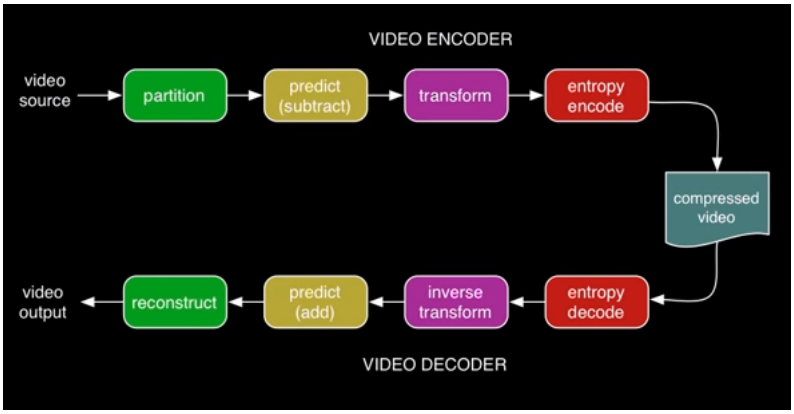

Figure 1.3: The steps of compressing and decompressing video a captured video

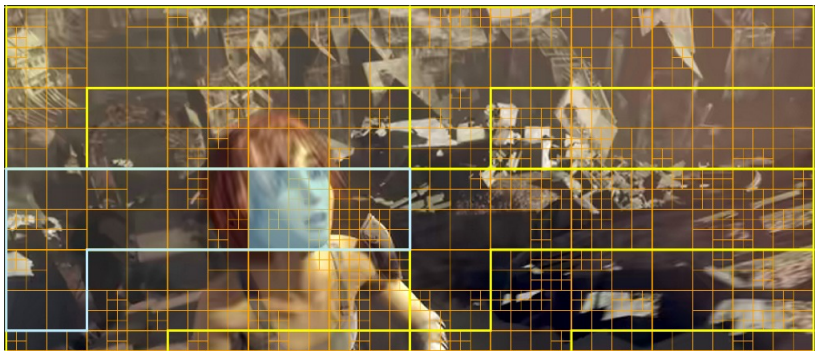

Figure 1.4: A slice outlined in blue, and quadtree splitting all across the frame

and each block can be split until it reaches the minimum block size of 8×8. [9]

Areas with sparse movement are easier to compress and will generally not be divided as much, forming larger continuous sheets in the quadtree structure. [2]

For color images with three components, each one is compressed individually. When chroma subsampling is used, this renders smaller transform matrices for the chroma components than for luma, but this is not a problem since encoders can handle transform block of differing sizes.

1.3.2 Prediction

To radically cut down on the information content, instead of working with the actual pixel values we attempt to generate a prediction of the frame, subtract it from the data and then encode this residual.

The prediction needs to be done in such a way that the decoder can reverse the process and recreate the image. This implies that only data that has been previously decoded can be used for further prediction, so the encoder has to follow the same rule. Although, of course, the encoder has access to the whole sequence in a unencoded form, it limits its predictions to what it has previously encoded to allow the content to be decoded. If the decoder cannot decode the data then the bitstream is worthless.

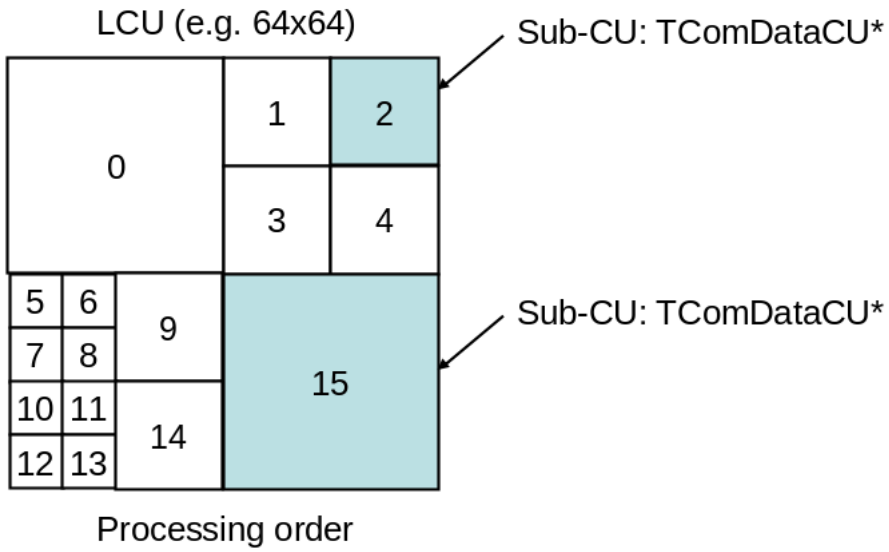

Figure 1.5: Quadtree splitting (from HM documentation)

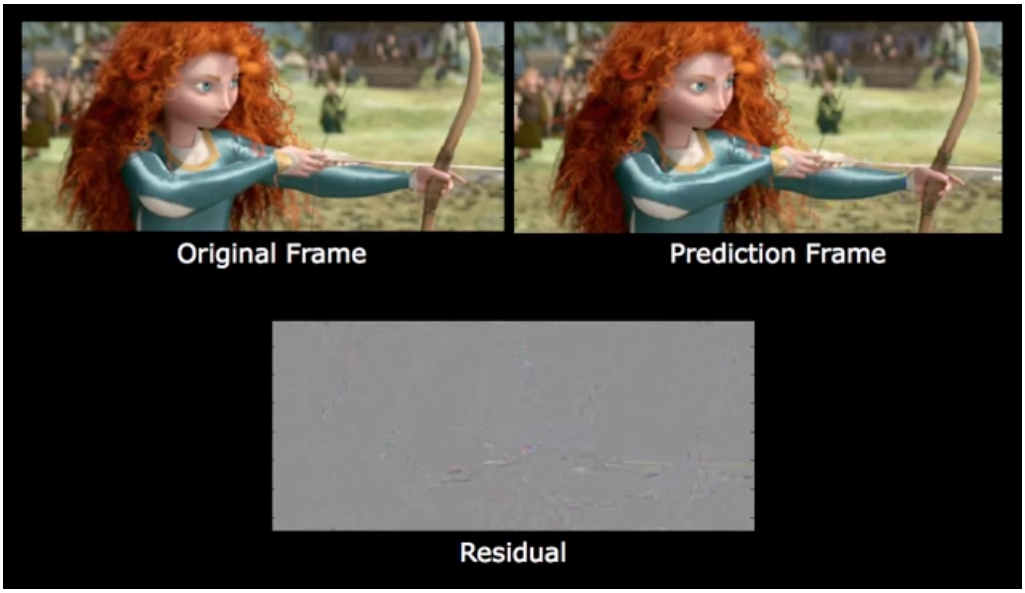

Figure 1.6: Prediction of a frame and the residual

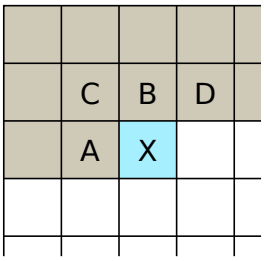

Figure 1.7: Intra prediction in JPEG

Intra Prediction

One prediction technique focuses on the internal relations between pixels in a images and is called *intra prediction*. Textures in non-random image tend to look like each other [10] – if you zoom in enough in any structure then a lot of pixels take the same color values – so it is assumed that fairly good predictions can be generated by approximating any pixel by its closest neighbors. Intra prediction as a concept is shared between image and video compression.

Figure 1.7 depicts the prediction of X using the values for neighboring samples A, B, C and D. Notice how no information from below or to the right of the image are used. The encoding moves row-wise from left to right, and those values have not yet been encoded.

Most of the information in the residual of the intra prediction will be contours of shapes in the picture, [4] as those are hardest to predict.

Inter Prediction

A more powerful prediction technique for video coding uses differences between frames, and is called *inter prediction*. The basic assumption is that objects in a frame will move around gradually, so if an object from a previous frame can be identified and followed, motion information can be used to make accurate predictions.

Identifying an object from one frame to another means that a search has to be performed. While it is not impossible to search the whole frame for the best match, for higher resolutions it is often very expensive to do so, so the search is generally confined to some nearby region. [10]

Using a mean squared error (MSE) calculation, [1] the closest match is identified, and a motion vector signals the displacement from one frame to the next. By storing only motion vectors, we avoid duplicating data from previous frames. [1]

The residual is calculated by taking pixel values of the referenced frame and shifting them according to the motion vectors. Then the prediction is subtracted from the actual frame, and this is what is passed on to the next step of the encoding. Again, we heavily cut down on the information content that we need encoded, but no information is irreversibly lost. Without the quantization step, the process would be fully reversible; the residual added back to the prediction and the effects of the motion vectors reversed.

When a sequence is encoded using inter coding, only certain frames or areas within frames will contain residual data for the actual picture content. Most of what we store is motion data together with references to one or several other frames. Thus, an inter coded frame cannot be decoded without first decoding the frames that it references. Depending

Figure 1.8: Motion vectors superimposed on the frame

on how good the prediction is – in a scene change, for example, it will generally be very bad – the residual will end up containing very little information.

As opposed to intra-prediction, contours of stationary objects will mostly disappear from this prediction, and what is left are those areas with hard-to-predict movements. [4] Noise from the original encoding can be misinterpreted as movement, like on the table in fig. 1.8. Although judging from the motion vectors in the rest of the image the table probably is not moving, the algorithm assigns big motion vectors to some its pixels which is unnecessary. However, because of the uniform nature to the surface, this won't hurt the prediction to much, although bigger motion vectors are generally more expensive to code.

The degree of accuracy in signaling the motion vectors comes at a certain cost. Every block of 16×16 coefficients gets one motion vector, with an accuracy ranging from one pixel to quarter-pixel, so the vertical and horizontal displacement could be modeled to one fourth's of a pixel's size. [2][5][1][10] More finely tuned motion-vectors cost more to encode, but may make up for it with better prediction, which produces smaller residuals – a typical compression trade-off.

It is up to the encoder to chose whether to intra or inter each section in a frame, [10] and this is done dynamically using techniques like rate-distortion optimization (see section 1.4.2).

["In order to achieve high compression efficiency, the encoder needs to try many different encoding parameters for each block (such as coding modes, block partitioning, motion vectors, etc.), a process that is sometimes referred to as mode/motion estimation and/or rate-distortion optimization (RDO). The process could be interpreted as a further step that precedes the prediction step."]

["Encoding is typically significantly more demanding than decoding in terms of computational complexity. The reason for that is that in order to achieve high compression efficiency, the encoder needs to try many different encoding parameters for each block (such as coding modes, block partitioning, motion vectors, etc.)." quote from patent i think, but verified here: Calculating motion vectors is the most expensive part of encoding, because of the wealth of options offered by HEVC. [9]]

1.3.3 Transform: DCT

A frame is divided into 16×16 [or 8×8?] blocks and frequency transformed using discrete cosine transform (DCT). [7] This separates high and low-frequency components toward the bottom right and top left of the transform matrix.

There exist a number of similar versions of DCT. Image and video compression generally uses DCT-II, shown in eq. (1.1), for calculating transform coefficients and DCT-III for inversing the process. [1]. This pair is thus often referred to as *the* DCT. [10]

$$X_k = \sum_{n=0}^{N-1} x_n \cos\left[\frac{\pi}{N}\left(n + \frac{1}{2}\right)k\right] \qquad k = 0, \ldots, N-1. \tag{1.1}$$

1.3.4 Quantization

The transformed coefficients are quantized (rounded) according to some parameter, statistically decreasing the amount of unique information within the matrix. [7]

The level of quantization – and the amount of degradation introduced – is controlled by a quantization parameter. A higher value indicates a harder compression, "flattening" the DCT matrix, bringing all values closer to the overall average.

By its nature, the DCT divides the coefficient by their frequency content. Low-frequency components in the top left, and higher-frequency toward the bottom-right. The low-frequency components are more visible to the human eye, so it is common to apply a quantization matrix that actively distorts higher-frequency components to a higher degree; basically low-pass filtering the matrix. [7]

The actual process of quantization is done by passing all values through a quantization function, that maps many values to the same one. A coarser quantization step makes this more likely to happen – the fewer quantization steps there are, the more numbers are going to be mapped to the same – which in turn produces a matrix that is cheaper to encode. This is the whole reason why quantization is performed.

Adjusting the quantization parameter (QP) value allows us to specify just how much degradation we are willing to accept in order to save bit rate – bitstream size is traded for visual accuracy. [7]

The quantization process introduces an irreversible distortion to the frame. [10] "Reversing" the process means that values end up at certain preset steps – they have been *quantized*.

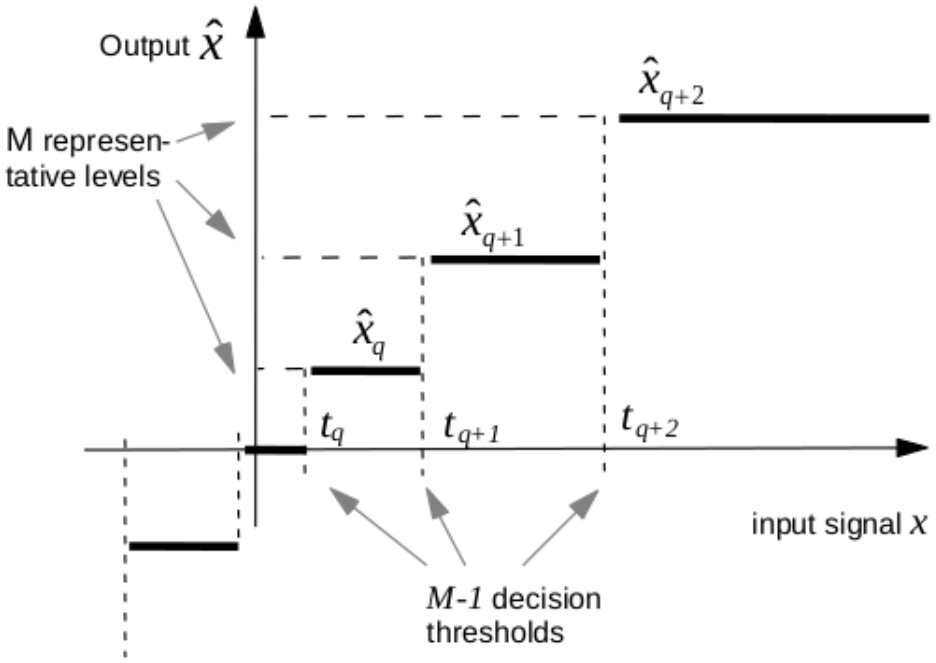

Figure 1.9: Quantization thresholds

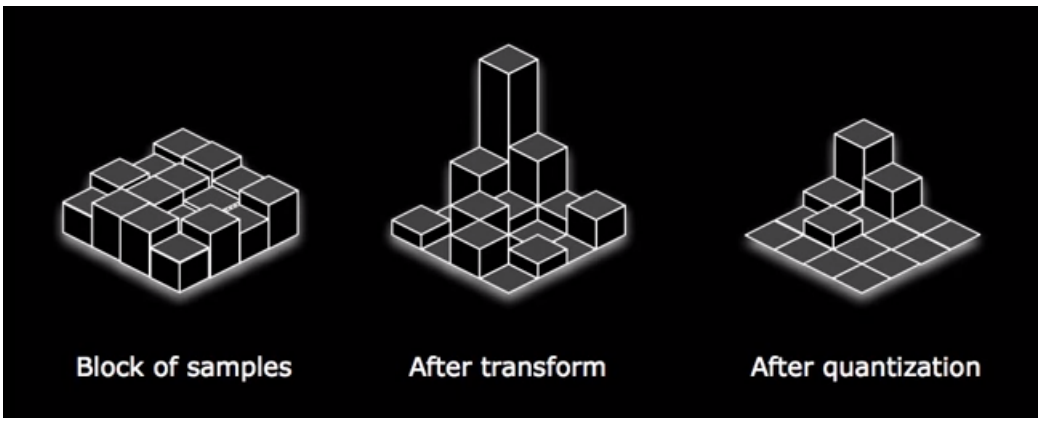

Figure 1.10: Quantization is performed in the transform domain

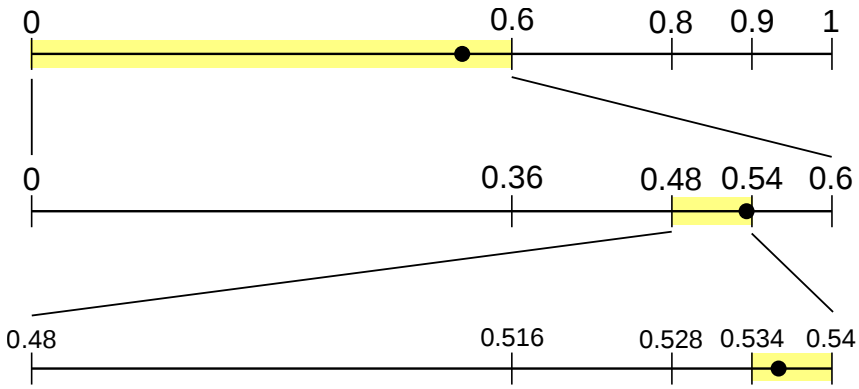

Figure 1.11: Arithmetic coding

1.3.5 Arithmetic Coding: CABAC

The reasons why quantization makes our files smaller is because we encode the sequences so that shorter numbers will take up less space. In the pixel domain, using 8 bits, each value take up the same amount of bits, so 0 for example is represented as 00000000. This is wasteful.

Arithmetic coding is a lossless compression method where a string of symbols are encoded as a fraction in the interval $I = [0.0, 1.0)$. By using the probabilities of each symbol and recursively subdividing I, we get a sub-interval uniquely representing our string.

By encoding the length of our original sequence and by then acquiring a binary fractional expansion residing completely within our interval, we can uniquely decode our message by reversing the process.

HEVC uses an algorithm called context-adaptive binary arithmetic coding (CABAC), [2] that was first introduced in H.264.

1.4 Video Coding

1.4.1 Lossy and Lossless Coding

By exploiting statistical redundancies within the images, information can be repackages as to allow perfect reconstruction. This is referred to as lossless encoding and means that no information is lost. [1] The basic idea is to represent information in a more efficient way. Huffman coding and CABAC are two examples of lossless coding.

Although the idea is attractive, the potential gains from only using lossless encoding are far to small to be effective, which is why lossy coding techniques are also employed. On top of a lossy algorithm, the compressed information is generally coded losslessly too, to further reduce bit rate.

Lossy coding techniques exploit not only statistical redundancies but also visual ones. Things that look similar in a frame can be coded together to save space. The aim of the process is to remove information that in not visible to the human eye, while leaving ev-

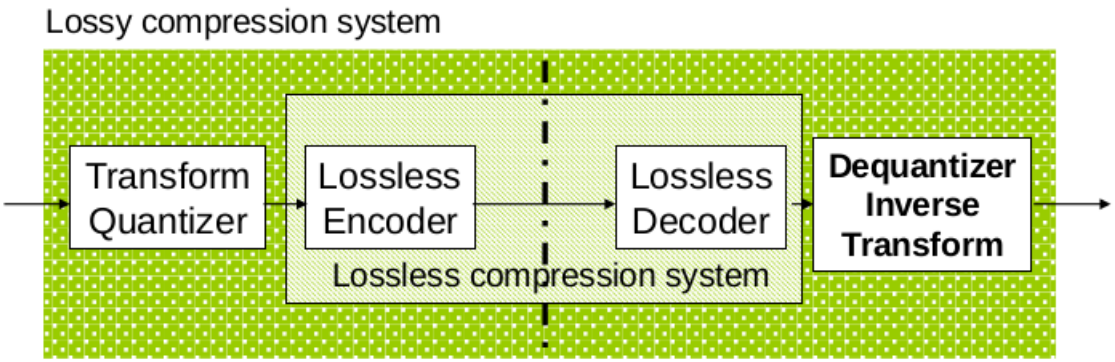

Figure 1.12: Lossless encoding within a lossy scheme

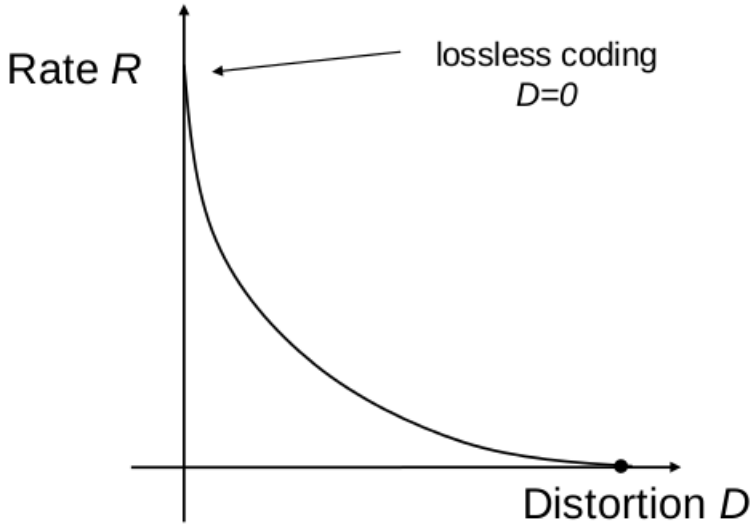

Figure 1.13: An RDO graph

erything that is visible. [1] This is a trade-off between quality and size. Depending on the application, images may be compressed to give visible artifacts and distortions, other times, a lossily compressed image looks near identical to its original.

1.4.2 Mode Decisions and Motion Estimation

Rate-distortion optimization (RDO)

To decide how to encode each block, intra or inter coding and other choices, RDO is often used. See eq. (1.2), where D stands for distortion and R stands for bit rate. λ is a weight parameter that can be assigned to steer the encoding in a certain direction.

$$J = D * \lambda + R \tag{1.2}$$

We encode each option and chose the one with the lowest J. That is, the choice with the lowest relative distortion to the rate.

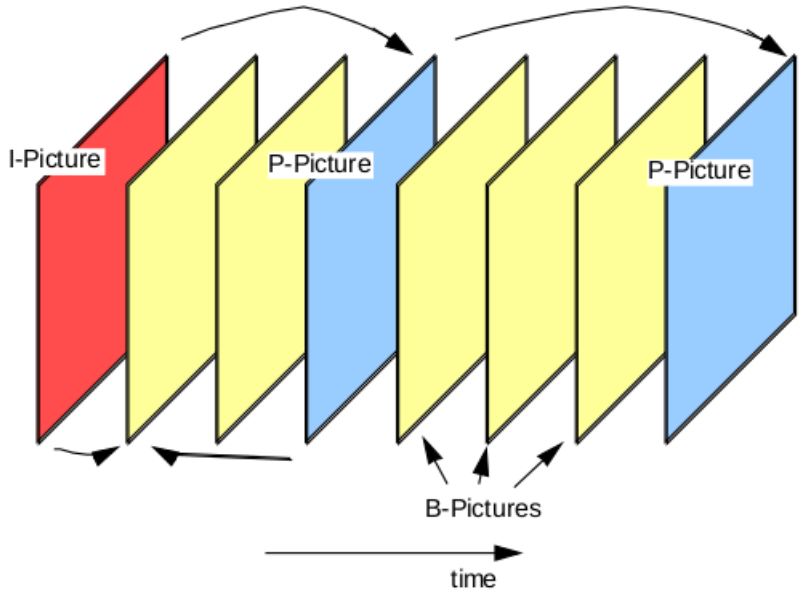

Figure 1.14: A group of picture containing three frame types

Out of a number of possible frames choices, the one with the lowest RDO value is generally chosen.

1.4.3 Types of Frames

Encoded frames are divided into categories based on how the interact with other frames. Intra coded frames are called I-frames. They appear at start of a sequence and at certain set points to allow the video to be decodable without having to start the process from the very beginning.

Most frames are predicted from other frames and are called P-frames or B-frames. A P-frames uses one other frame for its prediction and a B-frame use two, usually one before and one after itself.

To allow for a B-frame to use frames after itself in the sequences, frames are not encoded

1.4.4 GOPs and Temporal Layers

An HEVC encoder divides the input video into groups of pictures (GOPs) when encoding. A GOP is treated as a unit by the encoder [and can be decoded independently from the rest of the video.] This is what allows the viewer to skip ahead to a given time with a certain time interval without having to decode everything prior to it.

While larger GOPs generally give better encoding performance [reference to Johan], the ability to skip ahead in the video means that GOP sizes are chosen at least at every second or half-second. [actually intra period, which is 32 = 4 * GOP size for randomaccess]

We distinguish between output order, the order in which the material was captured, and the way it should be displayed, and the coding order, which is the encoders internal

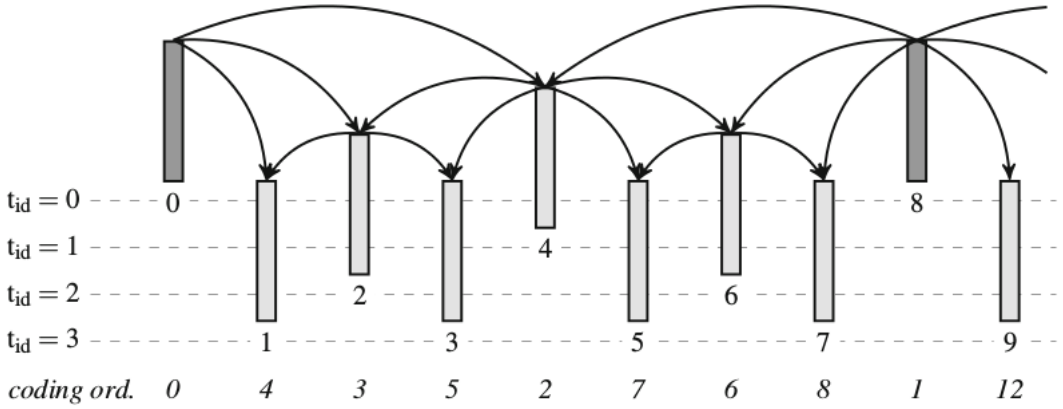

Figure 1.15: Temporal layers within a GOP

ordering. [10]

Inter-coded video is generally not encoded in the frame order that the material was captured in. The GOP structure specifies a coding order, [10] which is repeated cyclically throughout the encoding of a sequence.

We used the randomaccess GOP structure, which has a GOP size of 8 and, starting from index 0, has coding order 0-8-4-2-1-3-6-5-7, after which it skips to 16 and repeats the same pattern. [10]

Using a certain scheme, the first pictures in the GOP are intra coded by sheer necessity. They form temporal layer 0, because they depend on no other pictures. The following frames are mostly inter coded, and thus come to depend on the layer 0 pictures. These form layer 1. Pictures that depend on layers 0 and 1 becomes temporal layer 2, and so on. Randomaccess has a total of of 4 temporal layers.

The QP values are usually increased slightly for the higher temporal layers. The rationale is that they generally contain less data and so compressing these slightly harder won't have much of an effect on the sequences as a whole.

The coding order has implications of how prediction can be made. Because the decoder needs to do predictions in order to decode frames, and only previously decoded frames can be used to predict, no frame can depend on an yet unencoded frame. So although the coder has access to all frames, it has to restrict predictions – otherwise the video can not be decoded. [10]

A video of lower frame rate could be created by disregarding all frames above a certain temporal layer. [10] A similar idea was utilized by us for partial pruning in section 2.3.1.

1.5 History of Video Compression

Many of the basic concepts of video coding stem from the 1970s and '80s. [3]

[Inter and intra coding: Both of these techniques stem from the infancy [source] of video coding.] The changes between HEVC and the previous standard, AVC/H.264, are primarily in smalls tweaks that together account for the around 50% gains (either the same quality can be produced with half the size, or the same size produces video with "twice" the quality, or somewhere in between).

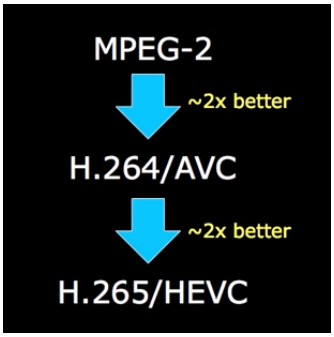

Figure 1.16: Video formats compared

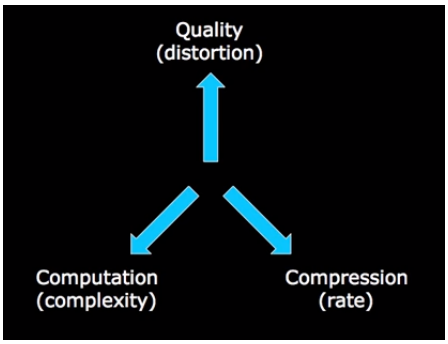

Figure 1.17: The fundamental trade-off in video coding

Compression keeps getting better partially because we have more computer power. We can perform deeper searches and find the best encoding schemes. [3] (See fig. 1.17)

The trend in video coding standards is for larger and larger block sizes to accommodate for larger resolutions. This is logical since HD (High-definition) content somewhat recently became a viable to store and transmit. While older standards, like H.263 supported only smaller block sizes, HEVC is more flexible and adapts block sizes depending on frame information. [2]

The Advanced Video Coding (AVC) standard was approved by Joint Video Team (JVT) in 2003 [10] and is in 2016 the most widely used format for HD video.

[..., AVC, HEVC. The main new features in HEVC compared to AVC]

1.6 HEVC-Specific Coding Algorithms

1.6.1 RDOQ

Rate-distortion optimized quantization (RDOQ) is a method of improving coding efficiency by manipulating coefficients after quantization. Out of the available quantization levels, we try different values and perform RDO to decide the best value to re-assign the coefficient.

Testing all possible combinations of quantization steps and coefficients out of a 16×16 matrix would very computationally heavy. Instead, we go through the matrix one-coefficient at a time from the bottom-right and backwards, setting it the all possible levels

or a subset of its closest neighbors.

1.6.2 Sign-Bit Hiding

Sign-bit hiding (SBH) is a lossy technique to decrease bit rate. For a 16×16 block of transform coefficients, to avoid transmitting the sign for the top-left element, it is instead inferred from the parity of sum of all coefficients.

[SBH is signaled in the sequence header and to tell the decoder calculate the parity and reconstruct the missing sign of the first coefficient]

Because transform coefficients are represented in base two, and all even base-two numbers end in 0 and all odds ones end in 1, only the last bit for each coefficient needs to be taken into account. Assuming that the sign of the coefficient in question is equally likely to be positive as negative and assuming that coefficients are equally likely to be even as odds, this parity is going yield the incorrect result half the time.

To counter this, the SBH algorithm performs RDO (how?) over the transform block to find the value that would have the least impact on the resultant image if changed up or down by one. That is, in order to save one bit on every transform block, a small error in introduced in half the blocks.

The result is typically well worth it. This is a typical lossy compression techniques, small losses in quality are traded for larger decreases in bit rate.

1.7 Adaptive Streaming

[Cloud platforms run on generic, non-specialized hardware. [6]]

["These different outputted video streams can have a lower spatial resolution, a lower temporal resolution, a lower quality, or even a different compression standard."]

Streamed video-on-demand content is usually delivered using adaptive bit rate (ABR) [6]. Depending on device capabilities and bandwidth, different users will request different quality content. A user that starts out watching a video when bandwidth usage is low will generally be sent high-quality video. If the available bandwidth then decreases – maybe by more users entering the network – the streaming service needs to dynamically lower quality to allow the video and alleviate the network.

[Transcoding is the conversion of one coded signal to another. [8] In the most straightforward case, this means full decoding and re-encoding a video. However, the process of encoding is expensive in terms of time, so we generally want to avoid it when possible.]

ABR means that the transmitted quality is adapted to the receiver's needs. Video cannot be downscaled in encoded form. Thus, to create a lower-resolution encoding of a video, it has to be decoded, downscaled and then re-encoded. This process is known as transcoding.

The encoding step generally takes a lot of time, so the allow adaptive streaming, the service creates encoding beforehand and store them until they are ready to be transmitted. This will naturally take up a certain amounts of space.

[discard the original]

[In a realistic scenario, a number or storages for original video and side-information are placed in urban areas. Hubs are then placed in large numbers, close to end-users. From the

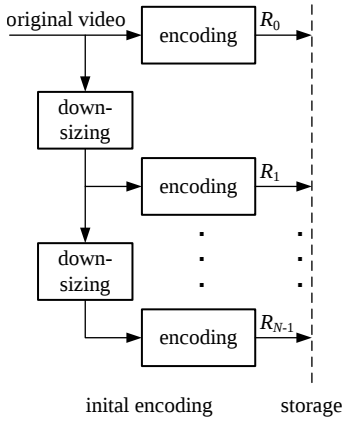

Figure 1.18: Simulcasting

storage areas, originals and side-information can be sent over to a hub at the first instance any user requests it, it is then stored at the hub for a certain amount of time depending of the demand, and then reconstructed on-the-fly when the end-user wants to watch the video.]

We present here three current models for adaptive streaming: simulcasting, just-in-time (JIT) transcoding and scalable video coding (SVC), and discuss their respective weaknesses. We then introduce *guided transcoding* as way to over-bridge many of these problems.

[We call the network connection between original storage and the hub performing the adaptation the uplink, and the connection between hub and end-user the downlink]

1.7.1 Simulcasting

The straightforward way create create adaptive content is that whenever a new video is added to the *adaptation node*, we scale the video to each desired size and encode. The original video in not kept on the adaption node. This model is known as *upfront transcoding* or *simulcasting*. [6]

A streaming service offering 1080p, 720p, 540p and 360p versions of all of its content will have one encoding for each resolution stored away and ready for transmission. This takes up a lot of space [9], and encodings are generated without prior knowledge of customer demand.

1.7.2 Just-in-Time Transcoding

The problem with simulcasting is that we may end up storing many files that are seldom or never actually requested. [6] A more attractive idea might then be to encode the original only at the highest resolution that we want to offer, and then transcode that to the lower-resolutions on demand. Lower-resolution encodings may then be as long as demand is high, and then throw and away. This model is known as JIT transcoding.

The general problem with JIT transcoding is that encoding usually takes a lot of time. Even with specialized hardware, the process of encoding thousands of frames is generally too slow for the idea to work in practice. [6][9] Unless encodings can be generated within

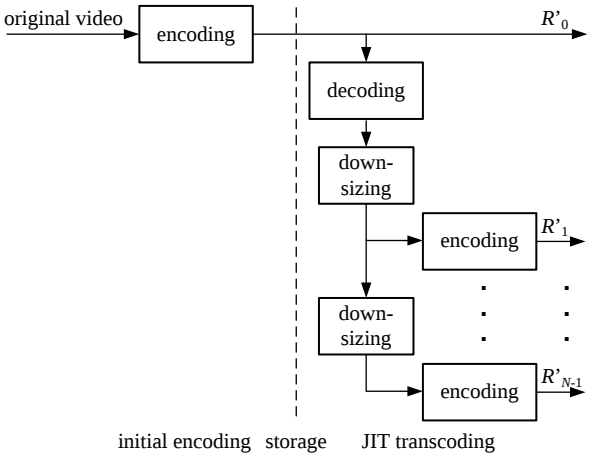

Figure 1.19: Just-in-time transcoding

minutes, JIT transcoding is not appropriate for video streaming. For example, the idea or "ordering" a movie at 540p and having to wait a day for the content to be available would likely scare away most costumers in a world where we already have proper on-demand streaming.

Another problem with JIT transcoding is that the lower-resolution content will be generated from an already-degraded data source. In simulcasting, each encoding is generated directly from the original, but now the highest-resolution encoding acts as an original. This problem is shared with our pruning approach to guided transcoding (GT) (see section 2.3).

This will likely introduce *generation loss*: coding artifacts and high-resolution noise that are amplified by successive transcoding, and lowers the video quality, [8] as well as unnecessarily increasing its size.

1.7.3 Scalable Video Coding

Scalable video coding (SVC) uses a layered model to encode different-quality representations of the same material. Starting with a base layer, which is regular encoding, one or several enhancement layers are encoded with reference to the base. Enhancement layers can only be decoded after first decoding the base layer.

Often, the different layers represent varying resolutions, with the base layer holding the lowest resolution, [source?] and each enhancement layer representing a higher resolution. Decoding a higher-resolution SVC layer requires access to the base layer together with every intermediate enhancement layer. Decoding an enhancement layer means the previous layer is decoded, upsampled to the size of the enhancement layer, and used for prediction. [source?] The process in repeated for every subsequent layer until you reach the target resolution.

[Find pyramid image]

The term SVC stems from the H.264 standard and is often known as SHVC (Scalable high efficiency video coding) in H.265, but we will use the terms interchangeably and refer to both as SVC.

The major problem with SVC is that the dependent structure causes a size overhead when encoding the enhancement layers. While storage space can be saved on the adapta-

tion node compared to simulcasting, any higher-resolution videos transmitted to the costumer will typically take up more space in SVC than in a regular encoding due to overhead from layering the content. [9]

Also, SVC generally performs better at the lower resolutions – for example, adding a new lowest-resolution encoding to the scheme moves every enhancement layer further away from the base and actually lowers coding efficiency. As such, the whole idea of SVC is in many ways out of step with the modern usage were the demand is on HD, Ultra HD, and beyond.

1.8 Previous Research

The idea of GT is not a novel one. [9] explores pruning as an idea and compares it to simulcasting, JIT transcoding and SVC for one low quality (LQ) representation. No mention is made of using partial techniques. We expand the experiments to large scale with the simulation environment we wrote, being able to test a multitude of sizes and shifting the QP up slightly, as well as using a work-around for finding accurate data on the role of SBH in pruning.

Deflation is an idea that we present for the first time in this thesis.

Chapter 2
Approach

2.1 Guided Transcoding

GT aims to get around some of the problems of the previously presented models for adaptive streaming (see section 1.7). Instead of transmitting full encodings of lower-resolution representations, we generate side-information (SI) which is cheaper to store, and then use the highest-resolution video together with the SI to quickly reconstruct the sequence using a specialized decoder. "Quick" is this scenario means that computational complexity of the reconstruction is comparable to that of regular decoding – that is, several orders of magnitude faster than doing an encoding.

Because the reconstruction process is a relatively fast – it can be done in real time under certain conditions – we can keep all lower-resolution videos stored only as side-information, and regenerate on-demand every time they are requested, thus achieving what was is generally not possible in the JIT transcoding scheme. Guided transcoding can be seen as a reversal of SVC, in that we save the highest-resolution video and use that regenerate lower-quality representations.

The lower-resolution video that we alter to generate the SI stems from the same original as the higher-resolution video, allowing us to perform perfect reconstruction. This is essential for the effectiveness of the method.

2.2 Transcoding Operations

2.2.1 Downscaling

The process of transforming a decoded video to a lower resolution is called downscaling or downsampling. The goal is to decrease the number of pixels in the image, lowering its quality and file size, while still preserving the overall "look" of each frame.

To decrease the number of pixels, groups of pixels are mapped to single values. To downscale by a factor of 2, for example, each 2×2 block is replaced by a single pixel. The downscaler looks at a number of adjacent pixels in the input video and calculates a weighted average to assign to the output. Each of the three components in a color image are downscaled separately, and the subsampling ratio (see section 1.2.5) is always preserved.

2.2.2 Decoding Video in Decoding Order

The reconstruction step in both GT schemes require the higher-resolution video to be in decoding order. This simply means that normal buffering of images, that is used for any non-sequential GOP structure is suspended. For the randomaccess scheme, this means that frames are output in the order []. Once again, the video can technically be displayed, but the frames are out of order.

2.2.3 Reconstruction

The reconstructor takes in two files, the SI and the higher-resolution video that the SI was generated from. The higher-resolution video must be in decoding order.

2.3 Pruning and Residual Reconstruction

The first model for generating SI is *pruning*. Pruning a bitstream means passing it through a modified decoder that removes all transform coefficients and replaces them with *sparse* dummy matrices that are cheap to encode, thus decreasing the file size. Left are the mode decisions and motion vectors – those that are the most expensive to calculate. [and even more so in the HEVC because it offers more dynamic options than previous standards. [6]]

By doing the encoding beforehand and storing the complex parts, we can reconstruct videos much faster than we would be to transcode or generate encodings. So the pruning achieves the virtues of both upfront and JIT transcoding; it saves space by allowing us not to store full encodings of every size, but is still fast so we can generate content on short notice.

[This is why we call it guided transcoding, or even guided JIT transcoding. The SI guides the transcoding process so to save complexity.]

The pruned bitstream is still decodable, but because all transform coefficients are removed (usually set to 0) the video will look far from normal. The screen shows contours of objects with what looks like very heavy-motion blur, and colors are distorted toward pink and purple (see fig. 2.1).

To allow for perfect reconstruction, the video that we prune must descend from the high quality (HQ) representation that will later be used in the reconstruction. We won't have access to the original video in the reconstruction step – storing it would be far to expensive – so we regards our highest-resolution video as an original and transcode it to generate LQ bitstreams. We encode LQ bitstreams with the same QP that the HQ bitstream was encoded with, or a slightly coarser one.

It the LQ bitstream that we prune, and will later reconstruct for adaptive streaming usage.

Figure 2.1: A decoded frame from a pruned bitstream

Because we transcode an already degraded bitstream – the HQ representation is a lossy encoding of the original – to generate the LQ bitstream, we introduce generation loss. This generally requires us encode the HQ bitstream with a slightly higher bit rate than in the simulcasting scenario to achieve the same quality for the LQ representations.

Reconstructing the video is very similar to the regular encoding process, except that all partitioning and mode decisions have already been done. We encode the pixel data from the decoded downscaled HQ video using mode decisions and the motion vectors from the SI.

2.3.1 Partial Pruning

To decrease the computational complexity of the guided transcoding even further, we experimented with the idea of *partial pruning*. That is, instead of removing all transform coefficients, we only prune certain frames, and then if we modify the reconstructor accordingly we can save complexity.

The partial pruning scheme is based on temporal layers and we designate different levels based on what is excluded from the pruning. Partial pruning level 1 means the the highest temporal layers, are excluded, level 2 excludes the top two layers, and so on. The Randomaccess GOP structure has a total of four levels, so we can do partial pruning level 1-3. Level 4 would mean no pruning, level 0 would be full pruning.

Higher temporal layers are P-frames and B-frames. They reference other frames for predictions and will generally contain fewer coefficients. Thus, we expect level 1 pruning to have a minimal effect of bit rate, compared to full pruning, while still allowing us to save computational complexity in the reconstruction.

2.4 Deflation and Inflation

Our second model for generating SI is to *deflate* the bitstream.

In this scheme we generate all encodings, HQ and LQ, directly from the original so there is no extra transcoding loss, like for the pruning scenario. A video is deflated by

Figure 2.2: A decoded frame from a partially pruned bitstream

downscaling the HQ encoding [and generating transform coefficients], these are then subtracted from the transform coefficients in the LQ video and the difference is stored. Because these two emanate from the same original, the differences in transform coefficients in expected to be small, and thus much cheaper to encode than each of the two input videos.

Inflation reverses the process. We decode and downscaled the HQ video and generate transform coefficients with are added back to the deflated coefficients to perfectly reconstruct the LQ bitstream.

We modified the reconstructor from the pruning scenario to turn it into a deflator and inflator. The HEVC standard does not allow coefficient matrices of all-zeros, so we identify all matrices of the form shown in eq. (2.1), and increase the N coefficient by one. `SHRT_MAX` is the highest possible value since we are storing the transform coefficients in variables of the type short.

$$\begin{pmatrix} N & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 \end{pmatrix}, \quad 0 \leq N < \texttt{SHRT\_MAX} \tag{2.1}$$

This scheme maps `SHRT_MAX` and `SHRT_MAX` $-$ 1 to the same value, and will potentially introduce a small distortion. But this values is highly unlikely to occur for this coefficient, the lowest-frequency one, which generally resides around 0 when calculating the difference between two similar videos. It has never appeared in any of our simulations.

In the inflation, we look for matrices of the form shown in eq. (2.2) and subtract every N by one. [Getting a resultant matrix of all-zeros is not a problem at this point, since we later add the coefficients from the HQ video.]

$$\begin{pmatrix} N & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 \end{pmatrix}, \quad N \geq 1 \tag{2.2}$$

Chapter 3
Evaluation

3.1 Experimental Setup

3.1.1 The Guided Transcoding Chain

Our test environment for running guided transcoding simulations has many components. In order to test different encoder settings, SBH on or off, RDOQ on or off, and different GT schemes: pruning and deflation, and their corresponding partial versions, we needed a dynamic system were changing settings was as easy as flicking a switch.

Running a full GT simulation chain means encoding several test sequences at multiple QP values, decoding and downscaling each encoding to a number of different sizes, re-encoding, generating SI and finally reconstruct the sequences. Additionally, we need to measure bit rate and peak signal-to-noise ratio (PSNR) at several points in the simulation, and be able to access the data in a structured manner in order to make sense of the values and compare different simulations on equal terms.

The pruning and deflation scenarios differ in the way side-information is generated, but many parts of the chain works the same way using most of the same programs. We first describe the GT approach common to both methods and then elaborate on the details where they differ.

3.1.2 Encoding the Original

The first step is to encode the original test sequences. Our suite uses five HEVC sequences that belong to the JVT common test conditions, class B: Kimono, ParkScene, Cactus, BasketballDrive and BQTerrace. They represent a wide variety of possible use-cases for video encoding: varying degrees of movement, static camera and panning movements, and frame rates varying from 24 to 60 fps.


Figure 3.1: Still frames from our HEVC standard sequences

All of the sequences have a bit depth of 8 bits – color vales are represented in 2^8 distinct steps, ranging from 0 to 255 – and 1920×1080 resolution. [10] Each clip is ten seconds long, giving a total number of frames from 240 to 600, which JVT considers sufficient to get a qualified assessment of video quality while still keeping the required encoding complexity sufficiently low. [10]

All videos we work with are stored in progressive scan, meaning simply that each unencoded frame contains information about all of its pixels, as opposed to interlaces scan where images are split over two successive frames, each holding half the lines. We will follow the convention regarding scan modes and refer our resolutions as 1080p, 720p, etc., where we width of the frame is omitted.

Some of our sequences have a copyright frame as the last one, giving the total number of frames for a 60 fps video at 601. This is not expected to used for simulations, so we exclude it for all of our encodings and references, and it should not have any effect on our results.

For each original video we create four encoding with QP values 22, 26, 30 and 34. We refer to these as QP_{base}. All encoding and possible re-encoding are performed using HEVC Test Model (HM) (see **??**), with a configuration file specifying the *randomaccess* GOP structure, [10] intra period of 32 frames and *HEVC Main profile*.

All of the outputs of from this step are referred to as HQ bitstreams. Generating these are by far the most expensive part of any simulation in terms of computational complexity, ranging from 10-20 hours on our cluster environment [see section ...], so we store them away and always try to re-use old bitstreams simulation where each applicable test parameters is the same.

3.1.3 Re-Encoding and Pruning

For the pruning scenario, the next step is to decode the HQ bitstreams and downscale them to each of the LQ resolutions. The we re-encode with the same QP value *and* with $QP+2$.

That is, for a HQ bitstream generated with $QP = 22$, we re-encode with QP values 22 and 24. For HQ bitstreams generated with $QP = 26$ we re-encode with 26 and 28, and so on. The reason for this is that we expect the low-resolution videos to be less sensitive to a slight increase in QP, and this allows us to quantify the effect.

After re-encoding, we prune the LQ bitsteams to generate SI. Depending on a simulation parameter we either do partial or full pruning. The difference at this stage is just one input parameter to the pruner binary.

In an real real-world application we would then store the SI instead of the LQ bitsteam, saving a certain amount of storage space. Of course, for the simulation we keep both to make comparisons. We refer to the SI and the bitstreams as uplink data, and measure the rate reductions as in eq. (3.1).

$$\frac{\text{LQ bit rate} - \text{SI bit rate}}{\text{LQ bit rate}} \tag{3.1}$$

3.1.4 Downscaled Originals

We downscale each original file from 1080p to 720p, 540p and 360p – referred to as the LQ resolutions – for use as reference data when calculating PSNR, and we also encode each one at every applicable QP value, the $QP_{extended}$ (22, 24, 26, 28, 30, 32, 34 and 36) for bit rate comparisons.

The downscaled originals also acts as LQ bitstreams in the deflation scenario.

3.1.5 Deflation

We generate SI for deflation using HQ encodings and the corresponding encoding of a downscaled original for $QP_{extended}$.

3.1.6 Residual Reconstruction

The HQ bitstream together with the side-information is used to reconstruct videos. The reconstruction in both scenarios is a perfect process. This is, it reconstruct perfectly what was removed during pruning/deflation.

For each test case, we reconstruct the video to make sure everything works as expected – that the file can be decoded and looks normal. We measure bit rate and PSNR again, although by design we could just as well have re-used the data for the corresponding LQ bitstream.

3.1.7 Platform LSF and Directory Structure

For any simulation of five tests sequences, four QP values, three LQ resolutions and then two additional QP values we get a total of $5 \times 4 \times 3 \times 2 = 120$ test cases.

Each combination is submitted to a cluster system as a self-contained "job". The cluster allows for faster calculations than running locally, and allows hundreds of jobs to run in parallel without affecting performance. The cluster is shared among many users and a has its own scheduler.

One job will for example correspond to BasketballDrive, QP 22, 540p and QP 24, and will then only be concerned with the creation of the specific files needed for that test case.

Another script is responsible for evaluating simulation parameters and submiting each of the 120 jobs that make up on simulation. The jobs then work in parallel, sharing many of the same files, and together generate every combination of files needed for a full simulation. The script also creates a test_data file to keep track of the locations of the bit rate and psnr data files.

To allow many jobs to work in parallel and read and write from the same directory structure we implemented a locking system. Because we could not communicate through the LSF system we could not utilize traditional threading so we represented locks as empty files named like the file we want to lock and the extra file extension ".lock".

Each script is given its own tmp area where files are created, and then they all share the same storage area. Whenever we want to create a file, we attempt to lock that file in the storage area. If this lock creation fails because the lock file already exists we assume another job is creating that file, and we enter a sleep loop that exits only when the lock has been removed.

The execution works as follows, if two jobs both have BasketballDrive and QP 22, but differ in the downscaled resolution and second QP, the will both want to create and HQ bitstream of QP 22.

Assuming the file does not already exist from some previous simulation, one will acquire the lock first and starts creating the file in its tmp area. The other waits. After the file has been created it is moved over to the storage area and the lock removed. This way, no in-progress or incomplete files are ever found in the storage area. And won't have to worry about race conditions in the chain. If the file is not locked and it exists in the storage area, it is guaranteed to be complete. Both jobs can then move over to the next step where they downscale the file and use it to generate other files.

Especially early on in the chain, many jobs share the same files. Out of 120 jobs, they are evenly divided into groups of 24 that each share the same HQ bitstream.

Every file in the chain follows the same pattern of creation. The jobs checks for the existence of the file in the storage area and if it find it there it moves on to the next step. If all files have already been created the the jobs runs through the whole chain without creating any files at all.

This allows for maximum file reuse on slightly different simulations. For example, when we run a simulation where some parameter only affecting the later part of the chain have been changed, each job will find many of its files already present and won't need to recreate them. In many cases we thus can avoid the encoding the originals, cutting down the total simulation time by several hours.

[graph of directory structure]

3.2 Evaluation Environment

3.2.1 Measuring Video Quality

The most common way to measure video quality objectively is to use PSNR. [10]

For two sequences emanating from the same source, usually an original and some degraded version, referred to as I and K, both with the same resolution $n\ timesm$, we calculate the difference per pixel and sum the across the whole frame to get the MSE. This is shown in eq. (3.2).

$$MSE = \frac{1}{m\,n} \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} [I(i,j) - K(i,j)]^2 \tag{3.2}$$

The PSNR per frame is then calculated as in eq. (3.3), where Max_I is the maximum value of the intensity function, 255 for 8 bit images. To get get the PSNR for the whole sequence we take the average for each frame.

PSNR is calculated separately for the luma and chroma components. We capture data for all three, but generally we are only interested in the luma component.

$$PSNR = 10 \cdot \log_{10}\left(\frac{Max_I^2}{MSE}\right) \tag{3.3}$$

The more similar two sequences are – the less distorted the encoded version is – the higher the PSNR value. PSNR as a number carries no significance in itself, but a higher PSNR is better, so the relative difference between two encodings is meaningful.

We always measure PSNR against the original. Each HQ bistream is decoded and compared with the full-size original. The LQ encodings are compared with downscaled originals.

3.2.2 Measuring Bit Rate

The bit rate is an absolute measure of the size per second of content. It is often presented in kilobits per second (kbit/s).

We use the file size of each encoding to calculate an average across the whole sequence. Because file sizes in regular operating systems are generally presented in bytes, we get the number of bits as in eq. (3.4). To calculate the bit rate (in kbit/s) we use eq. (3.5).

$$\text{bits} = \text{file size} \cdot 8 \tag{3.4}$$

$$\text{bit rate} = \frac{\text{bits} \cdot \text{framerate}}{\text{frames} \cdot 1000} \tag{3.5}$$

The bit rate is calculated for each bitstream. The LQ and HQ bitstreams, encoded downscaled originals and the SI.

To accommodate for the loss in quality caused by re-encoding in the pruning scenario, a third-degree polynomial is used to interpolate the theoretical bit rate required in order to achieve the quality of the the non-transcoded one. This way, bit rates can be compared on "equal terms".

3.2.3 Interpolation

[New interpolation formula, as a math formula]

3.2.4 Excel sheets

All the test data (bit rates and PSNR) are stored away in text files next to the files themselves. Utilizing the test_data file created by the job script, and the Python library *openpyxl* library we extract all the data from one simulation and structure it an Excels files.

The allows us to calculate average bit rates and PSNR for each sequence and for the whole simulation, to easily compare different simulation parameters. It also plots the data as scatter graphs.

Each Excel document has several sheets that interact with each other. "Originals" contains information for all sequences at all sizes of encodings of original sequences. These have the highest possible PSNR values but also the highest bit rates.

[picture of an originals sheet]

The uplink cost at any given size is the SI and the HQ bitsteam. The Uplink sheet together with Originals make up "Compare uplink" which contains comparisons between simulcasting and our GT case. Gains represent the difference in bit rate between the SI and the encoding we would otherwise store. The rate reductions show how much storage space can be saved.

[picture of compare uplink sheet]

The Downlink sheet deals with the communication between adaptation node and receiver. In the pruning scenario, the interpolation formula provides us a larger bit rate for the streamed video to compensate for the degradation of the re-encoding.

We measure costs as the extra bit rate we transmit to the receiver when doing GT. Where we saved bit rate in storing the video, we now pay for this by a PSNR reduction. To accommodate for this we interpolate the bit rate expected in order to give the same quality as in the simulcast case and then use this higher bit rate as a comparison.

The deflation case, because we can encode the original straight away, and thus transmit the same video to the receiver as in the simulcast case, no extra cost incurred at this step.

[picture of compare downlink sheet]

3.3 Results

One simulation means running the full chain of 120 test cases for any given setting of test parameters; RDOQ and SBH settings, pruning or deflation and their respective partial levels.

3.3.1 Pruning

Our residual reconstructor cannot handle SBH. Therefore we have to turn it off in the LQ encoding before before we prune to allow the pruned video to be reconstructed.

However, SBH generally produces better results, so we want it to be turned on when possible. To measure its effects, we generate LQ encodings with SBH on and prune. We then utilize the fact that the reconstruction is a perfect process – this is asserted programmatically every time a pruned file is reconstructed in the regular scenario – and use the bit rate and PSNR data for that file instead. This way we are can present data for a simulation we were not actually able to run.

The max gain column calculates the theoretical maximum that could be gained by a GT scheme where the SI is reduced to 0. Naturally, no scheme can achieve this, but I gives an indication of the effectiveness of our current methods.

Table 3.1: 160309_1915_5t_sbh0_pruning_rdoq0_fp_sao_off

Average bitrates									
	QP_{base}				$QP_{base}+2$			Totals	
	1080p	**720p**	**540p**	**360p**	**720p**	**540p**	**360p**	**GT gain**	**Max gain**
Kimono	2419	1043	760	475	835	593	362	33.9%	62.7%
ParkScene	3552	1341	888	472	1053	678	354	26.9%	57.7%
Cactus	8283	2799	1933	1118	2242	1522	868	28.3%	58.9%
BasketballDrive	8058	2852	2031	1207	2300	1605	938	26.7%	60.0%
BQTerrace	13605	2585	1718	889	1949	1295	664	25.2%	48.6%
Averages	**7183**	**2124**	**1466**	**832**	**1676**	**1139**	**637**	**28.2%**	**57.6%**

Rate reductions							
	QP_{base}			$QP_{base}+2$			Totals
	720p	**540p**	**360p**	**720p**	**540p**	**360p**	
Kimono	55.7%	54.0%	52.6%	55.2%	52.7%	50.2%	53.4%
ParkScene	46.0%	47.5%	50.6%	45.3%	46.0%	47.7%	47.2%
Cactus	48.4%	49.4%	52.3%	47.2%	47.4%	49.1%	48.9%
BasketballDrive	44.9%	45.5%	48.1%	43.8%	43.6%	44.9%	45.1%
BQTerrace	52.7%	53.6%	56.1%	52.2%	52.3%	53.8%	53.5%
Averages	**49.5%**	**50.0%**	**51.9%**	**48.8%**	**48.4%**	**49.1%**	**49.6%**

Average costs							
	QP_{base}			$QP_{base}+2$			Totals
	720p	**540p**	**360p**	**720p**	**540p**	**360p**	
Kimono	17.5%	10.1%	4.5%	11.4%	6.7%	2.9%	8.8%
ParkScene	15.6%	8.5%	4.0%	10.1%	5.4%	2.5%	7.7%
Cactus	14.8%	8.0%	3.6%	9.4%	5.1%	2.3%	7.2%
BasketballDrive	16.3%	9.2%	4.4%	10.3%	5.9%	2.7%	8.1%
BQTerrace	17.2%	9.1%	3.9%	11.3%	5.9%	2.6%	8.3%
Averages	**16.3%**	**9.0%**	**4.1%**	**10.5%**	**5.8%**	**2.6%**	**8.0%**

Average reconstruction speed (frames per second)							
	720p		**540p**		**360p**		**Totals**
	QP	**QP+2**	**QP**	**QP+2**	**QP**	**QP+2**	
qp 22/24	12.5	13.0	15.0	15.4	14.7	15.0	14.3
qp 26/28	16.7	17.0	20.1	20.6	19.2	19.4	18.8
qp 30/32	19.8	19.9	23.9	24.3	22.2	22.2	22.1
qp 34/36	21.7	22.3	26.7	26.8	24.2	24.2	24.3
Averages	**17.7**	**18.1**	**21.4**	**21.8**	**20.1**	**20.2**	**19.9**

Table 3.2: 160309_1939_5t_sbh0_pruning_rdoq0_pp_l1_sao_off

Average bitrates									
	QP_{base}				QP_{base} + 2			Totals	
	1080p	**720p**	**540p**	**360p**	**720p**	**540p**	**360p**	**GT gain**	**Max gain**
Kimono	2419	1043	760	475	835	593	362	32.3%	62.7%
ParkScene	3552	1341	888	472	1053	678	354	26.3%	57.7%
Cactus	8283	2799	1933	1118	2242	1522	868	26.9%	58.9%
BasketballDrive	8058	2852	2031	1207	2300	1605	938	24.5%	60.0%
BQTerrace	13605	2585	1718	889	1949	1295	664	25.0%	48.6%
Averages	**7183**	**2124**	**1466**	**832**	**1676**	**1139**	**637**	**27.0%**	**57.6%**

Rate reductions							
	QP_{base}			QP_{base} + 2			Totals
	720p	**540p**	**360p**	**720p**	**540p**	**360p**	
Kimono	52.1%	51.4%	50.9%	52.8%	51.0%	49.2%	51.2%
ParkScene	44.7%	46.4%	49.6%	44.5%	45.2%	47.0%	46.2%
Cactus	45.3%	46.6%	49.6%	44.9%	45.3%	47.2%	46.5%
BasketballDrive	39.8%	41.4%	44.7%	40.4%	40.9%	42.7%	41.6%
BQTerrace	51.9%	52.7%	55.3%	51.8%	51.8%	53.4%	52.8%
Averages	**46.8%**	**47.7%**	**50.0%**	**46.9%**	**46.8%**	**47.9%**	**47.7%**

Average costs							
	QP_{base}			QP_{base} + 2			Totals
	720p	**540p**	**360p**	**720p**	**540p**	**360p**	
Kimono	17.5%	10.1%	4.5%	11.4%	6.7%	2.9%	8.8%
ParkScene	15.6%	8.5%	4.0%	10.1%	5.4%	2.5%	7.7%
Cactus	14.8%	8.0%	3.6%	9.4%	5.1%	2.3%	7.2%
BasketballDrive	16.3%	9.2%	4.4%	10.3%	5.9%	2.7%	8.1%
BQTerrace	17.2%	9.1%	3.9%	11.3%	5.9%	2.6%	8.3%
Averages	**16.3%**	**9.0%**	**4.1%**	**10.5%**	**5.8%**	**2.6%**	**8.0%**

Average reconstruction speed (frames per second)							
	720p		**540p**		**360p**		**Totals**
	QP	**QP+2**	**QP**	**QP+2**	**QP**	**QP+2**	
qp 22/24	19.0	19.8	22.8	22.8	23.5	23.1	21.8
qp 26/28	26.3	27.3	32.3	32.1	32.0	31.5	30.3
qp 30/32	32.4	33.7	39.1	39.2	38.4	38.3	36.9
qp 34/36	37.6	38.6	45.0	45.7	43.5	43.6	42.3
Averages	**28.8**	**29.9**	**34.8**	**35.0**	**34.3**	**34.1**	**32.8**

Table 3.3: 160309_1939_5t_sbh0_pruning_rdoq0_pp_l2_sao_off

Average bitrates									
	QP_{base}				$QP_{base}+2$			Totals	
	1080p	**720p**	**540p**	**360p**	**720p**	**540p**	**360p**	**GT gain**	**Max gain**
Kimono	2419	1043	760	475	835	593	362	26.9%	62.7%
ParkScene	3552	1341	888	472	1053	678	354	24.2%	57.7%
Cactus	8283	2799	1933	1118	2242	1522	868	23.3%	58.9%
BasketballDrive	8058	2852	2031	1207	2300	1605	938	18.6%	60.0%
BQTerrace	13605	2585	1718	889	1949	1295	664	23.8%	48.6%
Averages	**7183**	**2124**	**1466**	**832**	**1676**	**1139**	**637**	**23.4%**	**57.6%**

Rate reductions							
	QP_{base}			$QP_{base}+2$			Totals
	720p	**540p**	**360p**	**720p**	**540p**	**360p**	
Kimono	41.2%	42.6%	43.9%	43.9%	44.0%	43.8%	43.2%
ParkScene	40.1%	42.5%	45.9%	41.1%	42.4%	44.5%	42.7%
Cactus	38.2%	40.1%	43.3%	39.1%	40.0%	42.1%	40.4%
BasketballDrive	27.9%	31.1%	35.2%	30.5%	32.2%	35.0%	32.0%
BQTerrace	48.0%	49.1%	51.9%	49.5%	49.5%	51.3%	49.9%
Averages	**39.1%**	**41.1%**	**44.0%**	**40.8%**	**41.6%**	**43.3%**	**41.7%**

Average costs							
	QP_{base}			$QP_{base}+2$			Totals
	720p	**540p**	**360p**	**720p**	**540p**	**360p**	
Kimono	17.5%	10.1%	4.5%	11.4%	6.7%	2.9%	8.8%
ParkScene	15.6%	8.5%	4.0%	10.1%	5.4%	2.5%	7.7%
Cactus	14.8%	8.0%	3.6%	9.4%	5.1%	2.3%	7.2%
BasketballDrive	16.3%	9.2%	4.4%	10.3%	5.9%	2.7%	8.1%
BQTerrace	17.2%	9.1%	3.9%	11.3%	5.9%	2.6%	8.3%
Averages	**16.3%**	**9.0%**	**4.1%**	**10.5%**	**5.8%**	**2.6%**	**8.0%**

Average reconstruction speed (frames per second)							
	720p		**540p**		**360p**		**Totals**
	QP	**QP+2**	**QP**	**QP+2**	**QP**	**QP+2**	
qp 22/24	30.4	31.5	35.9	36.7	37.4	37.8	35.0
qp 26/28	41.3	41.9	49.2	50.1	50.5	51.3	47.4
qp 30/32	48.8	52.3	60.3	62.2	61.5	61.6	57.8
qp 34/36	59.1	61.8	72.1	72.2	71.6	71.7	68.1
Averages	**44.9**	**46.9**	**54.4**	**55.3**	**55.2**	**55.6**	**52.1**

Table 3.4: 160309_1939_5t_sbh0_pruning_rdoq0_pp_l3_sao_off

Average bitrates									
	QP_{base}				QP_{base} + 2			Totals	
	1080p	**720p**	**540p**	**360p**	**720p**	**540p**	**360p**	**GT gain**	**Max gain**
Kimono	2419	1043	760	475	835	593	362	20.8%	62.7%
ParkScene	3552	1341	888	472	1053	678	354	21.4%	57.7%
Cactus	8283	2799	1933	1118	2242	1522	868	19.2%	58.9%
BasketballDrive	8058	2852	2031	1207	2300	1605	938	12.6%	60.0%
BQTerrace	13605	2585	1718	889	1949	1295	664	22.2%	48.6%
Averages	**7183**	**2124**	**1466**	**832**	**1676**	**1139**	**637**	**19.3%**	**57.6%**

Rate reductions							
	QP_{base}			QP_{base} + 2			Totals
	720p	**540p**	**360p**	**720p**	**540p**	**360p**	
Kimono	29.4%	32.7%	35.6%	33.8%	35.4%	36.8%	34.0%
ParkScene	34.4%	37.4%	41.0%	36.7%	38.4%	40.7%	38.1%
Cactus	30.0%	32.8%	36.2%	32.3%	33.8%	36.1%	33.5%
BasketballDrive	16.5%	21.0%	25.4%	20.4%	23.2%	26.3%	22.1%
BQTerrace	43.2%	44.8%	48.0%	46.3%	46.5%	48.7%	46.3%
Averages	**30.7%**	**33.7%**	**37.2%**	**33.9%**	**35.5%**	**37.7%**	**34.8%**

Average costs							
	QP_{base}			QP_{base} + 2			Totals
	720p	**540p**	**360p**	**720p**	**540p**	**360p**	
Kimono	17.5%	10.1%	4.5%	11.4%	6.7%	2.9%	8.8%
ParkScene	15.6%	8.5%	4.0%	10.1%	5.4%	2.5%	7.7%
Cactus	14.8%	8.0%	3.6%	9.4%	5.1%	2.3%	7.2%
BasketballDrive	16.3%	9.2%	4.4%	10.3%	5.9%	2.7%	8.1%
BQTerrace	17.2%	9.1%	3.9%	11.3%	5.9%	2.6%	8.3%
Averages	**16.3%**	**9.0%**	**4.1%**	**10.5%**	**5.8%**	**2.6%**	**8.0%**

Average reconstruction speed (frames per second)							
	720p		**540p**		**360p**		**Totals**
	QP	**QP+2**	**QP**	**QP+2**	**QP**	**QP+2**	
qp 22/24	49.4	50.3	57.9	58.4	61.0	62.0	56.5
qp 26/28	63.4	65.0	75.6	76.4	78.9	79.2	73.1
qp 30/32	76.9	78.9	92.4	93.6	94.3	96.1	88.7
qp 34/36	90.9	94.3	108.9	111.7	111.0	112.3	104.9
Averages	**70.2**	**72.1**	**83.7**	**85.0**	**86.3**	**87.4**	**80.8**

Table 3.5: 160309_1959_5t_sbh1_pruning_rdoq0_fp_sao_off

Average bitrates									
	QP_{base}				QP_{base} + 2			Totals	
	1080p	**720p**	**540p**	**360p**	**720p**	**540p**	**360p**	**GT gain**	**Max gain**
Kimono	2409	1058	769	477	846	599	363	35.5%	63.0%
ParkScene	3530	1353	893	472	1061	682	354	28.1%	58.0%
Cactus	8275	2823	1944	1119	2259	1527	868	29.4%	59.2%
BasketballDrive	8078	2886	2048	1212	2323	1615	941	27.8%	60.2%
BQTerrace	13791	2626	1728	887	1971	1297	661	25.8%	48.6%
Averages	**7217**	**2149**	**1476**	**834**	**1692**	**1144**	**638**	**29.3%**	**57.8%**

Rate reductions							
	QP_{base}			QP_{base} + 2			Totals
	720p	**540p**	**360p**	**720p**	**540p**	**360p**	
Kimono	58.2%	56.4%	55.0%	57.4%	55.1%	52.5%	55.8%
ParkScene	48.1%	49.1%	51.7%	47.4%	47.5%	48.8%	48.8%
Cactus	50.4%	51.1%	53.4%	49.1%	49.0%	50.4%	50.6%
BasketballDrive	46.8%	47.2%	49.4%	45.7%	45.3%	46.2%	46.8%
BQTerrace	54.0%	54.7%	56.8%	53.6%	53.2%	54.3%	54.4%
Averages	**51.5%**	**51.7%**	**53.2%**	**50.6%**	**50.0%**	**50.4%**	**51.3%**

Average costs							
	QP_{base}			QP_{base} + 2			Totals
	720p	**540p**	**360p**	**720p**	**540p**	**360p**	
Kimono	17.0%	9.7%	4.4%	11.0%	6.3%	2.6%	8.5%
ParkScene	15.3%	8.2%	3.8%	9.7%	5.1%	2.4%	7.4%
Cactus	14.3%	7.6%	3.5%	9.0%	4.8%	2.1%	6.9%
BasketballDrive	15.8%	8.8%	4.1%	9.9%	5.5%	2.5%	7.8%
BQTerrace	16.9%	8.8%	3.7%	10.8%	5.7%	2.4%	8.1%
Averages	**15.9%**	**8.6%**	**3.9%**	**10.1%**	**5.5%**	**2.4%**	**7.7%**

Table 3.6: 160309_1959_5t_sbh1_pruning_rdoq0_pp_l1_sao_off

Average bitrates									
	QP_{base}				QP_{base} + 2			Totals	
	1080p	**720p**	**540p**	**360p**	**720p**	**540p**	**360p**	**GT gain**	**Max gain**
Kimono	2409	1058	769	477	846	599	363	33.9%	63.0%
ParkScene	3530	1353	893	472	1061	682	354	27.5%	58.0%
Cactus	8275	2823	1944	1119	2259	1527	868	27.9%	59.2%
BasketballDrive	8078	2886	2048	1212	2323	1615	941	25.6%	60.2%
BQTerrace	13791	2626	1728	887	1971	1297	661	25.5%	48.6%
Averages	**7217**	**2149**	**1476**	**834**	**1692**	**1144**	**638**	**28.1%**	**57.8%**

Rate reductions							
	QP_{base}			QP_{base} + 2			Totals
	720p	**540p**	**360p**	**720p**	**540p**	**360p**	
Kimono	54.5%	53.7%	53.1%	54.9%	53.3%	51.4%	53.5%
ParkScene	46.7%	47.9%	50.6%	46.5%	46.8%	48.1%	47.8%
Cactus	47.3%	48.2%	50.6%	46.7%	46.9%	48.3%	48.0%
BasketballDrive	41.7%	43.0%	45.9%	42.2%	42.4%	44.0%	43.2%
BQTerrace	53.2%	53.7%	55.9%	53.1%	52.7%	54.0%	53.8%
Averages	**48.7%**	**49.3%**	**51.2%**	**48.7%**	**48.4%**	**49.2%**	**49.2%**

Average costs							
	QP_{base}			QP_{base} + 2			Totals
	720p	**540p**	**360p**	**720p**	**540p**	**360p**	
Kimono	17.0%	9.7%	4.4%	11.0%	6.3%	2.6%	8.5%
ParkScene	15.3%	8.2%	3.8%	9.7%	5.1%	2.4%	7.4%
Cactus	14.3%	7.6%	3.5%	9.0%	4.8%	2.1%	6.9%
BasketballDrive	15.8%	8.8%	4.1%	9.9%	5.5%	2.5%	7.8%
BQTerrace	16.9%	8.8%	3.7%	10.8%	5.7%	2.4%	8.1%
Averages	**15.9%**	**8.6%**	**3.9%**	**10.1%**	**5.5%**	**2.4%**	**7.7%**

Table 3.7: 160309_1959_5t_sbh1_pruning_rdoq0_pp_l2_sao_off

Average bitrates									
	QP_{base}				QP_{base} + 2			Totals	
	1080p	**720p**	**540p**	**360p**	**720p**	**540p**	**360p**	**GT gain**	**Max gain**
Kimono	2409	1058	769	477	846	599	363	28.3%	63.0%
ParkScene	3530	1353	893	472	1061	682	354	25.2%	58.0%
Cactus	8275	2823	1944	1119	2259	1527	868	24.3%	59.2%
BasketballDrive	8078	2886	2048	1212	2323	1615	941	19.4%	60.2%
BQTerrace	13791	2626	1728	887	1971	1297	661	24.3%	48.6%
Averages	**7217**	**2149**	**1476**	**834**	**1692**	**1144**	**638**	**24.3%**	**57.8%**

Rate reductions							
	QP_{base}			QP_{base} + 2			Totals
	720p	**540p**	**360p**	**720p**	**540p**	**360p**	
Kimono	43.2%	44.6%	45.8%	45.7%	45.9%	45.7%	45.1%
ParkScene	42.0%	43.9%	46.9%	43.0%	43.8%	45.4%	44.2%
Cactus	39.9%	41.5%	44.2%	40.8%	41.3%	43.1%	41.8%
BasketballDrive	29.4%	32.4%	36.1%	31.9%	33.5%	35.9%	33.2%
BQTerrace	49.2%	50.0%	52.5%	50.8%	50.4%	51.8%	50.8%
Averages	**40.7%**	**42.5%**	**45.1%**	**42.4%**	**43.0%**	**44.4%**	**43.0%**

Average costs							
	QP_{base}			QP_{base} + 2			Totals
	720p	**540p**	**360p**	**720p**	**540p**	**360p**	
Kimono	17.0%	9.7%	4.4%	11.0%	6.3%	2.6%	8.5%
ParkScene	15.3%	8.2%	3.8%	9.7%	5.1%	2.4%	7.4%
Cactus	14.3%	7.6%	3.5%	9.0%	4.8%	2.1%	6.9%
BasketballDrive	15.8%	8.8%	4.1%	9.9%	5.5%	2.5%	7.8%
BQTerrace	16.9%	8.8%	3.7%	10.8%	5.7%	2.4%	8.1%
Averages	**15.9%**	**8.6%**	**3.9%**	**10.1%**	**5.5%**	**2.4%**	**7.7%**

Table 3.8: 160309_1959_5t_sbh1_pruning_rdoq0_pp_l3_sao_off

Average bitrates									
	QP_{base}				QP_{base} + 2			Totals	
	1080p	**720p**	**540p**	**360p**	**720p**	**540p**	**360p**	**GT gain**	**Max gain**
Kimono	2409	1058	769	477	846	599	363	21.9%	63.0%
ParkScene	3530	1353	893	472	1061	682	354	22.3%	58.0%
Cactus	8275	2823	1944	1119	2259	1527	868	20.0%	59.2%
BasketballDrive	8078	2886	2048	1212	2323	1615	941	13.3%	60.2%
BQTerrace	13791	2626	1728	887	1971	1297	661	22.7%	48.6%
Averages	**7217**	**2149**	**1476**	**834**	**1692**	**1144**	**638**	**20.0%**	**57.8%**

Rate reductions							
	QP_{base}			QP_{base} + 2			Totals
	720p	**540p**	**360p**	**720p**	**540p**	**360p**	
Kimono	31.0%	34.3%	37.1%	35.2%	37.0%	38.5%	35.5%
ParkScene	36.0%	38.7%	41.8%	38.4%	39.7%	41.6%	39.4%
Cactus	31.4%	34.0%	36.9%	33.7%	34.9%	37.0%	34.7%
BasketballDrive	17.7%	21.9%	26.1%	21.6%	24.2%	27.1%	23.1%
BQTerrace	44.2%	45.6%	48.5%	47.5%	47.3%	49.2%	47.1%
Averages	**32.1%**	**34.9%**	**38.1%**	**35.3%**	**36.6%**	**38.7%**	**35.9%**

Average costs							
	QP_{base}			QP_{base} + 2			Totals
	720p	**540p**	**360p**	**720p**	**540p**	**360p**	
Kimono	17.0%	9.7%	4.4%	11.0%	6.3%	2.6%	8.5%
ParkScene	15.3%	8.2%	3.8%	9.7%	5.1%	2.4%	7.4%
Cactus	14.3%	7.6%	3.5%	9.0%	4.8%	2.1%	6.9%
BasketballDrive	15.8%	8.8%	4.1%	9.9%	5.5%	2.5%	7.8%
BQTerrace	16.9%	8.8%	3.7%	10.8%	5.7%	2.4%	8.1%
Averages	**15.9%**	**8.6%**	**3.9%**	**10.1%**	**5.5%**	**2.4%**	**7.7%**

Time measurements for reconstruction

We measured the time for the reconstruction for the all four options of pruning, full pruning, and partial pruning levels 1-3.

Guided transcoding aims a saving time, allowing video to be reconstructed on-demand. Side-information together with HQ bitstreams are generated ahead of time. Three parts of the process are to be done repeatedly when the need arises: decode in decoding order, downscale and reconstruction.

For the scheme to work realistically, these three steps should be done is real-time or at least close to real time as the end-user waits. We measured the time it takes to reconstruct pruned [and deflated] video for each QP value and downscaling size:

[data table, fps should ideally match or be higher than that of video. If we can only to 30 fps, then this schema is really only useful for a 30 fps video without waiting]

3.3.2 Deflation

We did not have time to implement RDOQ in the deflator's internal quantization. Thus, having RDOQ turned on for the original coding introduces a mismatch between the two videos giving a significant drop in the uplink gains.

Because of this, we always turn RDOQ off for the encodings in the deflation scenario.

[RDOQ gives mismatch and worse results, the only fair comparison between two methods should be sbh=0, rdoq=0. Use this to report differences]

Table 3.9: 160310_1703_5t_sbh0_deflation_rdoq0_fd

Average bitrates									
	QP_{base}				QP_{base} + 2			Totals	
	1080p	**720p**	**540p**	**360p**	**720p**	**540p**	**360p**	**GT gain**	**Max gain**
Kimono	2419	1407	927	536	1045	690	397	25.3%	67.5%
ParkScene	3552	1761	1056	518	1288	771	380	21.4%	62.4%
Cactus	8283	3557	2240	1212	2662	1694	920	22.2%	62.8%
BasketballDrive	8058	3647	2370	1315	2738	1791	997	20.8%	63.8%
BQTerrace	13605	3395	2000	955	2342	1439	699	20.7%	52.7%
Averages	**7183**	**2753**	**1719**	**907**	**2015**	**1277**	**679**	**22.1%**	**61.8%**

Rate reductions							
	QP_{base}			QP_{base} + 2			Totals
	720p	**540p**	**360p**	**720p**	**540p**	**360p**	
Kimono	34.9%	38.5%	40.1%	36.7%	39.2%	39.6%	38.2%
ParkScene	31.5%	35.8%	40.8%	32.5%	35.8%	39.6%	36.0%
Cactus	32.6%	37.1%	41.8%	33.5%	36.9%	40.5%	37.1%
BasketballDrive	30.1%	34.1%	38.3%	30.9%	33.8%	36.8%	34.0%
BQTerrace	36.7%	41.0%	46.0%	38.7%	41.6%	45.4%	41.6%
Averages	**33.2%**	**37.3%**	**41.4%**	**34.5%**	**37.5%**	**40.4%**	**37.4%**

Average costs							
	QP_{base}			QP_{base} + 2			Totals
	720p	**540p**	**360p**	**720p**	**540p**	**360p**	
Kimono	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
ParkScene	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
Cactus	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
BasketballDrive	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
BQTerrace	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
Averages	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**

Table 3.10: 160310_1704_5t_sbh0_deflation_rdoq1_fd

Average bitrates									
	QP_{base}				QP_{base} + 2			**Totals**	
	1080p	**720p**	**540p**	**360p**	**720p**	**540p**	**360p**	**GT gain**	**Max gain**
Kimono	2412	1423	940	545	1055	699	402	22.3%	67.8%
ParkScene	3630	1804	1082	533	1316	789	390	19.1%	62.4%
Cactus	8454	3571	2248	1224	2660	1696	927	19.0%	62.8%
BasketballDrive	8025	3603	2346	1314	2693	1768	993	16.5%	64.0%
BQTerrace	15280	3524	2038	969	2411	1463	709	18.1%	51.6%
Averages	**7560**	**2785**	**1731**	**917**	**2027**	**1283**	**684**	**19.0%**	**61.7%**

Rate reductions							
	QP_{base}			QP_{base} + 2			**Totals**
	720p	**540p**	**360p**	**720p**	**540p**	**360p**	
Kimono	32.3%	34.0%	33.8%	32.5%	33.2%	32.4%	33.0%
ParkScene	29.3%	31.7%	34.5%	29.3%	30.9%	32.7%	31.4%
Cactus	29.5%	31.9%	34.2%	29.2%	30.7%	32.3%	31.3%
BasketballDrive	25.0%	27.4%	29.4%	24.6%	25.9%	27.2%	26.6%
BQTerrace	33.8%	36.1%	38.7%	34.9%	36.1%	37.9%	36.3%
Averages	**30.0%**	**32.2%**	**34.1%**	**30.1%**	**31.4%**	**32.5%**	**31.7%**

Average costs							
	QP_{base}			QP_{base} + 2			**Totals**
	720p	**540p**	**360p**	**720p**	**540p**	**360p**	
Kimono	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
ParkScene	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
Cactus	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
BasketballDrive	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
BQTerrace	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
Averages	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**

Chapter 4
Conclusions

Both pruning and deflation can significantly lower the need for storage space in adaptive streaming applications.

Pruning has a higher rate reductions deflation, close to 30% although at the cost of some extra degradation that generally needs to be handled by increasing the bit rate that needs to be sent to the costumer.

Deflation has rate reductions just above 20%, but the scheme is without extra degradation.

[What about the fact that when video is reconstructed, we store both SI and reconstruction, increasing the need for storage space compared to simulcasting!]

Bibliography

[1] Markus Flierl. Image and Video Processing, KTH EQ2330. Lecture slides, 2015.

[2] Iain Richardson. HEVC: An Introduction to High Efficiency Video Coding.

[3] Iain Richardson. Introduction to Video Coding.

[4] Iain Richardson. HEVC Walkthrough, Jun 2013.

[5] Iain Richardson. Video Coding Walk-Through. Technical report, Vcodex, 2013.

[6] Thomas Rusert, Kenneth Andersson, Ruoyang Yu, and Harald Nordgren. Guided Just-in-Time Transcoding for Cloud-Based Video Platforms. Submitted to ICIP 2016, Jan 2016.

[7] Timothy Sauer. *Numerical Analysis*. Pearson, 2nd edition, 2012.

[8] Anthony Vetro, Charilaos Christopoulos, and Huifang Sun. Video Transcoding Architechtures and Techniques: An Overview. *IEEE Signal Processing Magazine*, March 2003.

[9] Glenn Van Wallendael, Jan De Cock, and Rik Van de Walle. Fast Transcoding For Video Delivery by Means of a Control Stream. In *19th IEEE International Conference on Image Processing (ICIP 2012)*. Ghent University – IBBT, ELIS Department - Multimedia Lab, 2012.

[10] Mathias Wien. *High Efficiency Video Coding*. Springer, 2015.