TECHNISCHE UNIVERSITEIT EINDHOVEN
Department of Mathematics and Computer Science

MASTER'S THESIS

Complexity Scalable Motion Estimation Control for H.264/AVC

by
E.A.M. Huijbers

Supervisors:

Dr. R.J. Bril (TU/e)
Dr. T. Özçelebi (TU/e)

Eindhoven, August 2009

Contents

Preface and acknowledgements

This thesis is the result of a nine month research project carried out as part of the Computer Science program at the Systems Architecture and Networking group (SAN) at the Eindhoven University of Technology (TU/e).

Although the context of this work has always been mobile videocommunication, the focus of my efforst have shifted through the research time from network-centric to processor-centric, and afterwards from the x264 video encoder to the JM reference encoder. Unfortunately, these shifts have caused me not to be able to explore all the areas of this interesting topic that I would have liked. I hope that future research will address these areas.

I would like to extend my gratitude to my supervisors, dr. Reinder Bril and dr. Tanır Özçelebi, for their continued insight and guidance throughout the project, and dr. Richard Verhoeven for providing me with access to any hardware that I needed. The SAN supercomputer has assuredly helped me in acquiring experimental results in a quantity and detail that would not have been otherwise possible. Without these people, this thesis would not have been possible.

I would also like to thank all my colleagues at the SAN department, both University employees and fellow master students, for the inspired discussions and frequent social events. They have made the time that I have spent there a great experience.

Summary

Video encoding is a process that transforms raw video data into a compressed format. It is used to reduce the storage and/or transmission requirements of video, as uncompressed video data takes up a lot of space. The encoding process requires a lot of resources in the form of working memory, computational power and still a fairly large amount of storage if you want to achieve good quality. These resource requirements are not fixed but depend on the contents of the video being encoded or decoded. This means that the exact resource requirements cannot be known beforehand and fluctuate over time. In an environment with restricted resource availability, which may also be fluctuating, the encoding of video therefore becomes challenging.

A solution to this problem is using a video encoder that is *scalable*, meaning that it can adapt its operation to function under some restricted set of resources. We distinguish between two types of scalability: *rate* scalability and *complexity* scalability. The goal of rate scalability is to have the encoder produce an output bitstream that will survive transmission across a channel of fluctuating capacity. The goal of complexity scalability is to enable the encoder to continue producing chunks of output within some deadline, even under fluctuating computational resources.

Complexity scalability is useful when the encoded video must be produced within some timeframe. If there are no such requirements, it does not matter how much (computational) time encoding takes. In that case, even if the amount of available computational resources changes there is no point in adapting to it.

An important goal of scalable video encoding is that at every point in time, we want to achieve the best *quality* possible for the amount of resources available at that time. Available resources must not go to waste, which means that we want to achieve fine-grained scalability. The encoder must be able to operate at any point in its scalability range (from minimum to maximum).

In this thesis, we study an H.264/AVC encoder operating at a fixed bitrate under an arbitrarily restriced amount of computational power and with a deadline on every frame, i.e., the video must be produced in real-time. This scenario is consistent with video conferencing on a device with limited computational power, such as a mobile phone.

We show that we can guarantee that the encoder does not miss a deadline for encoding a frame by introducing a complexity budget in the encoder, placing a hard upper bound on the computational requirements of a single frame. Our complexity budget limits the number of Sum of Absolute Differences (SAD) computations that are performed, which is the most expensive operation of the motion estimation process of video encoding.

To obtain the best video output quality for a given frame budget we introduce an algorithm that allocates a frame's complexity budget to its constituents, called "macroblocks", based on the distortion with the best reference block block found during motion estimation of the previous frame. A high distortion indicates a bad match and is a sign that the macroblock could benefit from extra steps of motion search. We show that this allocation strategy results in higher quality than both the trivial Uniform Allocation strategy and the Motion History Matrix (MHM) [1] allocation technique, which uses the motion vectors found during motion estimation to estimate the complexity needs. We achieve quality gains of around 1 dB Peak Signal-to-Noise Ratio (PSNR) at around 30 dB PSNR encoding the "city" sequence at 1500 kbps.

Chapter 1

Introduction

In which we describe the background and motivation for this thesis. It specifies our goal and discusses related work.

1.1 Context

Nowadays, video applications are becoming more and more popular. As applications such as video conferencing start to gain popularity, video processing is no longer the domain of high-end computing hardware but is also introduced into low-cost consumer electronics devices.

Many video applications have real-time requirements. For example, if a sequence of pictures is being captured from a camera, each picture must be encoded and stored before the next one is captured. If the encoded video has to be sent over a communication channel to a live viewer, as is the case in video conferencing, every picture must be encoded and transmitted at least at the rate that the decoded pictures should be shown at the receiver side. This means individual pictures, called "frames", have *deadlines*, which is a point in time at which the encoding must be finished.

Buffering can be used to loosen these requirements a little, but buffering will only counteract temporary fluctuations in throughput; if the input or output requirements of the video are structurally higher than what the encoder is able to keep up with, frames will be dropped.

Why is video encoding used? It takes a large amount of space to represent video data on a computer. Video consists of a rapid sequence of images, each image consisting of a number of image points, each of them having a color. The color of every image point in every image has to be stored, so the video can be reconstructed and displayed again at a later time. As you can imagine, this amount of data adds up quickly. Video encoding changes the representation of the uncompressed video data, so that the representation takes up a lot less space. Before the video stored in this representation can be displayed again, it has to be *decoded* first, which reconstructs the original images that can be sent to a display (see Figure 1.1).

Video encoding is a form of compression. Its goal is to reduce the size of video information so it can be more easily stored and distributed. Like most compression algorithms, it is computationally intensive, since a lot of computational effort goes into

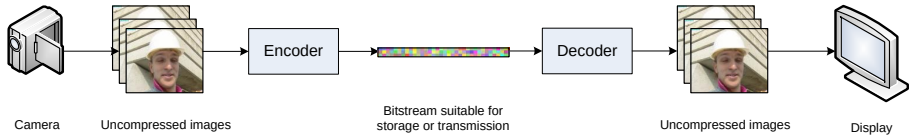


Figure 1.1: The place of video encoding and decoding. A sequence of images is captured from some image source. It is then encoded (compressed) to some bitstream format for long-term storage or transmission. Before the video can be displayed, the bitstream is decoded, after which the pictures can be sent to a display.

searching for efficient representations of the input pictures with the goal of having the bitstream be as small as possible.

Because Consumer Electronics (CE) devices have limited processing power and because the resource requirements of video processing are data-dependent, meaning the requirements cannot be predicted in advance but depend on the contents of video, it becomes a challenge to guarantee the real-time encoding or decoding of arbitrary video streams. Additionally, when the video processing happens on a mobile device, the battery drain is another factor to take into account, as a mobile device has a limited supply of power and power consumption is proportional to the amount of computation done. Available battery power places a limit on the amount of work that can be done before the battery is drained and the mobile device shuts down.

Since current video encoders and decoders are not prepared to deal with such resource limitations, device manufacturers have to redesign their applications for every device, increasing the production cost. Our goal is to design a video encoding system that continues to produce usable output under constrained resources and performs well on a variety of devices.

1.2 Motivation

In the past, network connectivity was typically the bottleneck for real-time video applications on mobile devices. A large amount of research has been carried out both into making the size of the encoder's output bitstream predictable (using *rate control* algorithms) and changing the format of the bitstream such that parts of it can be discarded without the stream becoming entirely undecodable, merely of degraded quality (using *rate scalability* methods).

These days however, most devices are able to use WiFi access points to connect at high speed to the Internet. Additionally, even the speeds of cellular data networks are steadily increasing, with the current standard UMTS capable of speeds of up to 21Mb/s. As the bottleneck for real-time video delivery to consumer electronics devices is no longer bandwidth, other bottlenecks become apparent, such as the restricted computational power available to low-end consumer products.

It is possible to cope with the problem of reduced processing power by limiting the video encoder and decoder for a given device to an operational mode that requires less processing power than the "full mode", such that even under the worst-case video load possible, the software is guaranteed to meet its deadlines. However, because of the necessity of assuming the worst-case load, the processing mode will be set too

pessimistically for the majority of the video content processed by the system. Since less processing also leads to lower quality, the system will not achieve the best possible video quality even though the resources to achieve high quality may be available. Our aim is to develop a system that does not have a limited set of operating points, but can operate smoothly along the entire range of processing power, from minimum to maximum.

In addition to the restricted processing mode yielding subpar quality in many cases, the act of finding out the settings appropriate for this mode itself is an act that has to be repeated by the manufacturer again and again for every kind of device they produce, as the performance characteristics will be different each time. This is an expensive and error-prone process for the manufacturer.

The solution to this is to develop software that can perform a video processing task, such as encoding a frame, under an arbitrary computational time constraint. This computational time constraint can then be selected to guarantee the needs of the system: it can be set high to achieve high quality, it can be set low to satisfy completion within a deadline or to conserve battery power. We call this limit on the computational resources the system is allowed to consume the *complexity budget*, and a system that can change the amount of processing power it consumes to stay within the budget *complexity scalable*.

Complexity scalability can be used to allow a system to function under restricted computational power, but it can also be used to scale back the resource consumption of the tasks in a system so the system can accept more tasks. Complexity scalable video encoding is useful in the following application areas:

- *Video encoding on resource-limited devices*. Low-end electronics devices have limited processing power, and there may be other processes (such as an audio encoder) that are contending for the processor in parallel with the video encoder. By being able to bound the processing requirements of the video encoder, we can ensure that our video encoder works unmodified on any hardware and software configuration simply by setting the appropriate bounds.

- *Video encoding on battery-powered devices*. Mobile electronics devices are powered by a battery. The battery power consumed is proportional to the amount of calculation that is performed, so by limiting the processing power consumed by the video encoder, we can extend the battery life of the device.

- *Video encoding in a system with hard real-time requirements*. Even though the processing power of the device may not be limited, it may need to be shared with a number of processes, each of which needs the guarantee of real-time operation. By placing an upper bound on the processing time of the video encoder, schedulability of the system can be guaranteed. In essence, every encoder can be guaranteed its own share of the system resources as a virtual platform.

- *Multiple video encoders accepting more work*. In the general case, a video transcoding server takes video in one format, decodes it and re-encodes it in a different format, usually before sending it off to some client to be displayed. Such a server may handle multiple clients at once and so may have to transcode multiple videos at the same time. When the video encoding processing is complexity scalable, the server may reduce the complexity of existing encoding tasks so it can accept more clients.

Complexity scalability has the following advantages:

- *Graceful degradation*. If the resources needed to do an optimal job are not available, the software can still accommodate the user by providing reduced (but still acceptable) service.
- *Homogeneous software*. A manufacturer of embedded systems can save time and money on software development for a variety of devices, if he can use a scalable solution that can make optimal use of the available resources on every device.
- *Real-time constraints*. In a real-time system, deadlines may be placed on a video encoding task. A complexity scalable encoding task can be made to meets its deadline by default, as the deadline can be translated into a complexity budget that the encoder can be constrained to.
- *Processor use corresponds to power consumption*. For mobile devices, it can be desirable to restrict a high-complexity task such as video encoding to preserve battery life.

In this thesis, we will focus on the video encoding side of video processing, as video encoding requires a lot more computational power than video decoding.

1.3 Goal

To control power consumption and to make guarantees about encoding deadlines, we want to be able to bound the complexity requirements of a video encoding process. Since a frame is typically the unit of video on which a deadline is imposed, we want to be able to bound the complexity consumption of a single frame.

We also want to have fine-grained scalability. That is, the encoder should be able to operate across the entire domain of complexity bounds. Operating in a fixed number of encoding modes would waste available complexity budget if that budget happens to fall just below the threshold of an operating mode.

Finally, reducing the complexity consumption of the encoder will also reduce the quality of the video. Our goal is to keep the distortion introduced by restricting the computational work to a minimum.

In this research, we will concentrate on the latest video coding standard jointly developed by ISO and ITU-T, called H.264/AVC [2]. H.264 has been adopted as Part 10 of the MPEG-4 standard and is also known as MPEG-4 Advanced Video Coding (AVC).

1.4 Related Work

Several ways to achieve complexity scalability of a video encoder have been introduced in literature, focusing mostly on the motion estimation part of encoding.

Our work is based on the work presented in [1] which provides a framework for modeling Rate-Distortion (R-D) performance as a function of power consumption for a

H.263+ video encoder, based on the scalability of three encoding parameters: the number of Motion Estimation (ME) operations, the number of Discrete Cosine Transform (DCT) computations, and the temporal resolution of the video.

In [3], H.264 is made complexity scalable by choosing one of four modes of motion estimation, of increasing complexity. Their work does not deal with allocating the frame budget among macroblocks, but does offer a way of optimizing mode decisions under restricted complexity.

In [4], complexity scalability is achieved by doing reduced motion estimation at the frame level, and reduced DCT at the macroblock level. The drawback is that their analysis requires access to all frames of the video before encoding begins (i.e. it is not suitable for a situation where the video is directly captured from a camera).

1.5 Contributions

In this thesis, we first extend the work of He *et al.* [1] into the domain of H.264 encoding. We apply their motion estimation scalability mechanism to the standard motion search algorithm in H.264 encoding, and extend it to cover the H.264-specific feature of inter mode partition decisions (see Section 3.2).

We then make a qualitative analysis of a number of algorithms designed to allocate the frame budget to macroblocks in such a way that quality is maximized. We introduce a new algorithm that estimates the complexity needs of a macroblock based on the distortion achieved during motion estimation. Our algorithm consistently leads to equal or higher quality output than the algorithm published in [1] (see Section 3.3).

1.6 Overview

The rest of this master thesis is organized as follows. Chapter 2 gives an overview of the aspects of H.264 video encoding that are relevant to this thesis. Chapter 3 describes the approach taken to make an H.264/AVC encoder complexity scalable and optimize quality. Chapter 4 presents our evaluation of the proposed algorithms, and finally Chapter 5 contains our conclusion and recommendations for future work. Appendix A contains a full list of results for all video sequences that we tested. Appendix B details some initial results about work that could not be completed withing the scope of this thesis. A glossary of terms is provided in Appendix C.

Chapter 2

Overview of Video Encoding

In which we present a high-level overview of video encoding. Specifically, the areas of a video encoder related to the rest of this thesis.

Video encoding is the act of compressing video data. But what does "video data" mean? Video consists of a number of pictures, called "frames", each taken at a regular time interval. The number of pictures taken per second is called the "frame rate". A typical frame rate is for example around 25 frames per second for European DVD video. Every frame consists of a number of image points called "pixels". For European DVD, a typical frame resolution is 720×576 pixels. The color of every pixel needs to be encoded, which in computer graphics is usually done by storing three bytes representing the intensity in a red, green and blue channel for each pixel. This means that in uncompressed form, a second of DVD video would take up $720 \cdot 576 \cdot 25 \cdot 3 = 31 \cdot 10^6$, or more than 30 megabytes. Since uncompressed video takes up such a large amount of space, it is expensive to store and impractical to transmit over a network. Even some harddrives may have trouble transferring data at this rate. The domain-specific compression techniques used in video encoding can reduce the size of the video data by a factor of 10 with very little loss of quality.

This chapter will give an overview of the techniques used in video encoding according to the H.264/AVC standard. The part of encoding related to coding analysis, specifically mode decisions and motion estimation, will be given the most treatment as these are the most relevant to the rest of this thesis.

2.1 Goal

A video encoder takes uncompressed video data and transforms its representation in such a way that the representation takes up a lot less space than the original data did. We will refer to this encoded representation as the encoder's "bitstream". The encoded bitstream cannot be displayed directly; first, it must be decoded by a piece of software called a "decoder" that reads the bitstream and reconstructs the original video data.

Video encoding is a kind of *compression*. There are two forms of compression: *lossless* compression and *lossy* compression. Lossless compression can work on any binary

data, and guarantees that the composition of the encoding and decoding operations yield the identity function. That is, decoded data is guaranteed to be exactly the same data that was encoded. Lossy compression on the other hand allows for some difference between the originally coded data and the reconstructed data. This allows the compression algorithm to achieve much better compression ratios than using lossless encoding, but it requires that some amount of information from the input data is discarded. This means lossy encoding is only applicable to some types of data such as audio and video that are intended for human consumption and can have some amount of information discarded without becoming incomprehensible, as opposed to things like executable files or text documents. It also means lossy compression algorithms have to be developed for a specific domain, because they require knowledge about the structure of the data they are operating on to decide what information to discard.

The goal of a video encoder is to encode a video signal to the smallest possible bitstream size, while keeping the difference between the decoded video and the originally encoded video small. In video encoding, the difference in the signal introduced by encoding, meaning pixels that have a different color value in the decoded version than in the encoded version, is called "distortion".

Distortion is typically measured in one of two metrics: Sum of Absolute Differences (SAD) or Sum of Squared Differences (SSD). It can be measured over various elements in a video: it can be measured over the entire video, over a single frame in the video or over a block of pixels in a frame in the video. Given two sets of X samples (pixel values) $A(x), B(x), 0 \leq x < X$, they are defined like this:

$$SAD(A,B) = \sum_{0 \leq x < X} | A(x) - B(x) | \tag{2.1}$$

$$SSD(A,B) = \sum_{0 \leq x < X} | A(x) - B(x) |^2 \tag{2.2}$$

A notion related to distortion is "quality". There are two kinds of quality, subjective quality and objective quality. Subjective quality is the quality as judged by a human watching the video. Ultimately, the goal of video encoding is to produce videos with a high subjective quality as the goal of video encoding is always to play the videos back to a human observer. Subjective quality is hard to quantify, however, which is why we typically measure objective quality. Objective quality is an attempt to quantify quality based on the differences between pixel values of the original and encoded video. Objective quality is historically measured in PSNR, which is derived from the SSD, and is defined like this:

$$PSNR(A,B) = 10\ log_{10}\ \frac{S^2}{\frac{1}{X} SSD(A,B)} \tag{2.3}$$

Where X is again the number of samples, and S is the maximum value of a sample; for normal 8-bit pixel values, $S = 255$. The terms distortion and (objective) quality refer to the same thing and can be used interchangeably, although for distortion, a lower value is better and for quality, a higher value is better.

The encoder's job is to optimize the bitstream for two metrics: to keep both the size of the bitstream (commonly called "rate") and distortion low. During encoding, a tradeoff exists between these metrics, as a decrease in the rate will lead to an increase in

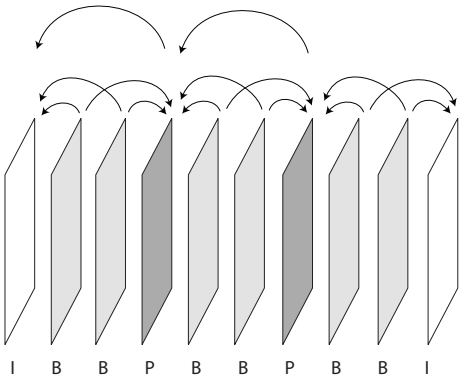


Figure 2.1: How different frame types refer to each other. Each arrow indicates a (possible) reference from a P or B-frame to an I or P-frame.

distortion, and vice-versa. When we compare encoders or encoder algorithms, we will evaluate them on their joint R-D performance. An encoder with a better R-D performance will be able to encode the same video at a lower rate with the same distortion, or will be able to reduce the distortion at the same rate.

The next sections describe how an H.264/AVC video encoder works.

2.2 Frames

The basic unit of a video is a frame. A video consists of a sequence of frames, shown in quick succession. MPEG-based encoders such as H.264/AVC can encode a frame in one of two ways:

- Intra-coded frames, usually called I-frames or keyframes. These contain enough information to be completely reconstructed by the decoder using only data in the encoded frame. They are robust, but need to contain a lot of information so they are very large.
- Inter-coded frames, usually called P-frames for *Predicted*, or B-frames for *Bi-directional*. These frames can refer to pieces of previously decoded frames, in lieu of encoding the pieces themselves. By referring to previously decoded samples, the bitstream can be made smaller because storing duplicate information can be avoided. Inter coded frames take up less space in the bitstream than intra-coded frames, but are less resilient against bitstream corruption and take more computational power to encode.

  P-frames only refer to I and P-frames that temporally precede the current frame, while B-frames refer to I and P-frames that both temporally precede and succeed the current frame. Note that the use of B-frames requires frames to be encoded and stored out-of-order. B-frames allow for a lot of R-D improvement, but require significant special treatment from the encoder to deal with the intricacies of forward-referencing frames.Other than that, they are not fundamentally different from P-frames, and the principles presented in this thesis can be easily extended to include B-frames, so we will not treat them specifically any further.

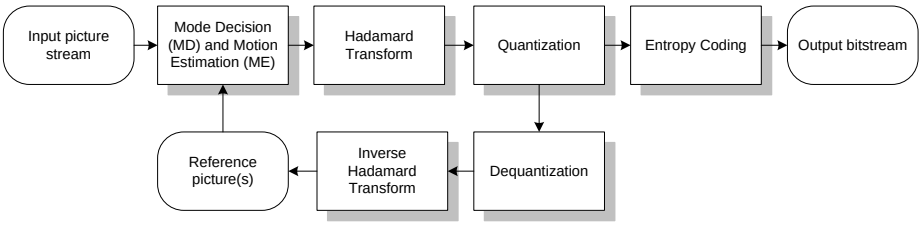


Figure 2.2: Data flow in an H.264 video encoder. The rectangles represent processing steps, while the rounded rectangles represent input, output and an important intermediate result.

See Figure 2.1 for a schematic view of how the different frame types depend on each other. For obvious reasons, the very first frame of an encoded video sequence must always be an I-frame, since there are no decoded frames yet to refer to. After that, an encoder is free to use inter-coded frames. Because long stretches of inter coding degrade the image quality, the encoder will usually insert a new I-frame at regular intervals to "refresh" the picture. The sequence of frames from one I-frame to the next is called a Group of Pictures (GOP).

The size of a GOP is at the encoder's discretion and can vary wildly: because I-frames take up a lot of space in the bitstream, the interval between keyframes can be as much as several hundred frames for streams where space-efficiency is the major concern, or a couple of dozens of frames if consistent quality is valued more. Many encoders will also try to detect a scene change in the video and start a new GOP at that point to establish a new frame of reference for motion estimation.

Although frames are the most recognizable building block of a video, they are not encoded in their entirety. For the purposes of encoding, frames are divided up into small units called *macroblocks*. Every macroblock is a block of 16×16 pixels, and is encoded and decoded separately. Upon decoding, the individual macroblocks are recombined to form the original frame again. Macroblocks are the fundamental unit of encoding. The next section describes how macroblocks are encoded in an H.264/AVC encoder.

2.3 H.264/AVC Video Coding

H.264 is a transform-based, block-based, motion compensated video format jointly developed by the International Standards Organization (ISO) Moving Pictures Expert Group (MPEG), and Telecommunication Standardization Sector (ITU-T). It has been adopted as Part 10 of the MPEG-4 specification, and is both known as H.264/AVC (ITU-T name) and MPEG-4 Part 10 or MPEG-4/AVC (ISO name). The flow of data in an H.264 encoder is shown in Figure 2.2, with all the processing that is done.

The basis of the video compression is formed by the Hadamard Transform, Quantization and Entropy Coding steps. These steps are excellently suited for the compression of natural images and are essentially the same as used in the JPEG still image coding standard. To this compression basis, Motion Estimation is added to improve the compressive qualities of the encoder for moving pictures. The extra processing steps,

those on the bottom row of Figure 2.2, are necessary to do effective Motion Estimation. Dequantization and the Inverse Hadamard Transform are actually decoding steps; every motion compensated video encoder also contains (part of) a decoder, so that the encoder can faithfully reconstruct the image as received by the decoder and use that as its *reference picture* for the motion estimation process.

The different parts of Figure 2.2 are explained below.

Input picture stream The input picture stream is a sequence of bitmap pictures. A single picture in the stream is called a frame, and the sequence is encoded on a frame-by-frame basis. A frame is a rectangle made up of of sampling points called pixels, each having a color value, and all frames have the same dimensions. Instead of the Red-Green-Blue (RGB) colorspace commonly used in computer graphics, MPEG-based video formats operate in the *YCbCr* colorspace, which means that every sample value is encoded in two orthogonal channels: a *luminance* channel (Y) and a *chrominance* channel (C). The chrominance consists of two coordinates (Cb and Cr) in a color plane. Because the human eye is much more sensitive to luminance than chrominance, the resolution of the chrominance channel is quartered before encoding. This subsampling already halves the size of the video data, at a slight loss of image fidelity.

In the rest of the thesis, when we talk about samples or pixels, we mean the pixel values in either the luminance (brightness) or one of the two chrominance (color) channels of the video. Each channel is compressed separately, and we will not explicitly refer to them any more. When we are evaluating quality however, we will only measure distortion between the luma channels of two videos. The chroma channel is much easier to encode and the human eye is much less sensitive to it, making the luma distortion the most important metric.

H.264 is a block-based format, which means that every frame is divided into blocks of 16×16 pixels in size, called macroblocks. Every block is encoded separately according to the steps specified here.

Mode Decision and Motion Estimation Mode decision and motion estimation are both analysis steps that are done before the actual encoding.

The goal of mode decision is to select one of several ways in which the macroblock can be encoded, so that the R-D performance is optimized. During mode decisions, a number of alternative ways to encode the block will be evaluated, after which the most optimal mode will be chosen. Mode decisions will be discussed in more detail in Section 2.5. As part of evaluating a mode, motion estimation may be done.

The goal of motion estimation, sometimes called motion compensation, is to exploit temporal redundancies in the input picture stream. By temporal redundancies, we mean blocks of pixel values that are the same or nearly the same in different frames. By describing the relative motion of the macroblocks of the previous frame with respect to the current frame, and only storing the pixel values that have changed, a lot of space can be saved in the bitstream as the unchanged pixel values do not have to encoded again. Because a frame that is motion estimated is not self-contained, motion estimation is only done for inter-coded frames, not intra-coded frames.

To find these redundant block of pixels, the encoder searches the reference pictures, which are previously encoded frames, for a pixel block of the same size and same

pixel values as the macroblock it is trying to encode. If a matching block can be found, only a pointer to the reference block needs to be sent to the encoder. As it is not always possible that there is a block in one of the reference pictures that *exactly* matches the current macroblock, it is allowed that there is some difference. The encoder then encodes the pointer to the reference block, and the difference between the reference block and the current block. The smaller this additional difference is, the less space it will take up in the bitstream, and the more efficient the compression is.

Motion estimation is a computationally expensive step, because any pixel position in any reference frame is a potential match for the current macroblock, so in principle all possible points need to be evaluated. In practice, fast motion estimation algorithms employ heuristics to drastically cut down the number of search points.

Note that although in older video coding formats motion estimation was always performed against the previous frame, in H.264 the encoder is free to use up to 16 frames as reference frames. Using more reference frames increases the memory requirements for both encoder and decoder since all reference frames have to be kept in memory and increases the complexity requirements of motion estimation by a lot since every reference frame has to be searched. The advantage is that the chances of finding a matching block are increased.

Motion estimation is discussed in more detail in Section 2.6.

Hadamard Transform The next step in a transform-based encoder such as H.264 is to transform the representation of the macroblock — either the macroblock pixel values or the residual after motion estimation — from their sample values into the frequency domain of a set of waveform functions of varying frequency, similar to a Discrete Fourier Transform. In H.264, Walsh functions are used, which repeatedly take either the value -1 or 1 at some frequency.

The result of this operation is a coefficient matrix for a predefined set of functions of varying frequency. In this representation, correlations between adjacent pixels from a natural image source can be represented with fewer bits than if they had to be stored directly. This step is said to exploit "spatial redundancy".

H.264 uses 2 different 4×4 matrices and a 2×2 matrix for the transform operation. Which of these matrices is applied is based on the type of macroblock that is encoded. The precise details of the transform operation are out of the scope of this work, but the details can be found in [5].

It remains to note that the transform is lossless; it merely changes the representation of the input values, as an input for the next step.

Quantization Quantization is the operation that provides the actual compression. In quantization, all values in the transform coefficient matrix of the Hadamard Transform are divided by a constant Q, and the result is rounded down to the nearest integer. This reduces the absolute value of the coefficients, which is done because the next step (entropy coding) is more efficient at storing numbers with small absolute values. Q can be chosen by the encoder for each individual macroblock. The larger Q, the better the compression and the smaller the resulting bitstream, but the more information will be lost due to rounding errors.

Entropy Coding In the last stage, all data that needs to be encoded for a macroblock, such as the displacement of the reference block, quantized transform coefficients and the value of Q, are stored and encoded using a lossless Variable Length Coding (VLC) compression algorithm. In H.264, the allowed algorithms are Context-Adaptive Variable Length Coding (CAVLC) and Context-Adaptive Binary Arithmetic Coding (CABAC).

The bitstream that is the result of this step is the encoded video stream that will be stored or transmitted to the decoder.

Dequantization Dequantization is the inverse operation to quantization, and simply multiplies all the coefficients of the transform matrix by the Q parameter. The result of this step is an approximation of the original coefficients prior to the quantization step, depending on the value of Q chosen by the encoder.

Inverse Hadamard Transform Finally, to produce a picture in the spatial domain again, the dequantized coefficient matrix is multiplied with the inverse transform matrix. This step and the previous one are the exact same as performed by the video decoder, and the result is the same picture as decoded by the decoder.

2.4 Rate Control

The video encoder can be run in one of two modes: *fixed quantizer* or *fixed rate*. The difference lies in how the quantization parameter Q is chosen, which is the parameter that controls the quality versus compression trade-off. The higher Q, the better the compression but the more information is lost and the more quality will degrade.

Fixed quantizer mode means that the quantization parameter Q that is used during the quantization step is set to a fixed value. This does not imply that Q needs to be constant across all macroblocks; Q can be chosen differently for intra-coded blocks and inter-coded blocks, I-frames and P-frames, but the values for Q are chosen before encoding starts and remain the same throughout. In this encoding mode the rate, i.e. the size of the output bitstream, can not be controlled but depends on the contents of the video that needs to be encoded, and how well motion estimation is able to find matching macroblocks. The larger the contrast and the finer the detail in the video, the higher the rate.

When a video is encoded in fixed rate mode, some target rate is supplied to the encoder. The encoder will start to encode the video with some initial value for Q, measure the rate of the bitstream, and adjust the Q parameter to try to match the supplied rate. If the bitstream turns out to have lower rate than allowed, Q is lowered, and the other way around, if the bitstream is too large, Q is increased. The encoder will continue monitoring the rate and keep adjusting Q to match the desired bitrate. In this mode, the quality is variable depending on the contents of the scene. The finer the detail and the higher the contrast, the higher Q will need to be to achieve the desired rate, so the more information will be lost and the lower the quality of the resulting video.

Any change to the video encoder that allows it to encode the same amount of information with less bits improves the performance of the encoder in either mode. In fixed

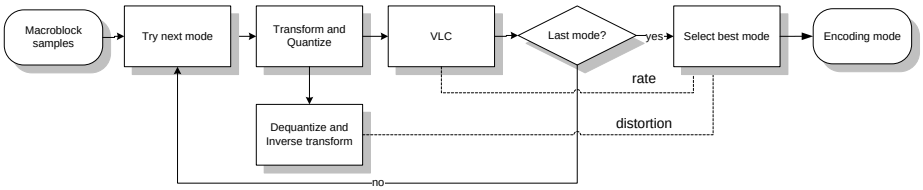


Figure 2.3: Mode decision algorithm for a macroblock. The macroblock is encoded in all possible modes, and the most optimal one is stored in the final bitstream.

quantizer mode, such a change directly leads to a lower rate. In fixed rate mode, the change allows Q to be lower to achieve the same rate, so better quality is achieved. For example, the closer the blocks found during motion estimation matching the current macroblock, the less coefficients still need to be encoded for the current block and the better the R-D performance of the encoder.

In the following sections, we will describe the analysis step, where mode decision and motion estimation is done, in more detail, as those are most relevant for the remainder of this thesis.

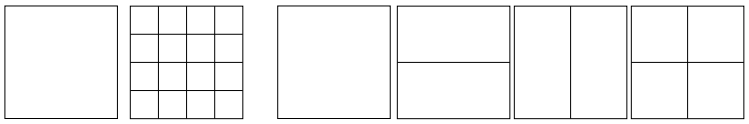
2.5 Mode Decisions

The encoder is free to encode every macroblock in one of several modes. There are several *intra* modes, and several *inter* modes. Similar to intra and inter-coded frames, intra-coded blocks can be decoded by themselves, while inter-coded blocks refer to blocks of pixels in a reference frame. If the current frame is an I-frame only intra modes are allowed, otherwise both intra and inter modes may be chosen. This means that it is possible that blocks can be intra-coded even in an inter-coded frame; this can be useful if no good block match can be found during motion estimation, to avoid having to encode a large difference.

When a macroblock is encoded in an intra mode, just its quantized coefficient matrix is stored, along with the quantization parameter Q. When a block is encoded in an inter mode, the index of the reference frame that it refers to is stored along with a motion vector that indicates the displacement from the macroblock's natural position to the position in the reference frame, in addition to the coefficients and the quantization parameter.

Actually, H.264 has an additional feature called intra prediction, which allows intra-coded blocks to use the sample values of previously decoded blocks in the same frame to predict the values of the current block, in a way similar to motion estimation. Though predicted intra blocks cannot be completely decoded by themselves, they only depend on blocks from the current frame, and this feature allows for extra size reduction of the bitstream.

Every encoding mode of the macroblock will result in a different number of bits stored in the bitstream, and a different distortion when reconstructed. The process of selecting the best mode to encode a macroblock in is called "mode decision", and is shown graphically in Figure 2.3. For every mode μ, the resultant rate ($R(\mu)$) and distortion ($D(\mu)$) are calculated, and finally the best mode is chosen by minimizing the Lagrangian cost function:



(a) Intra (b) Inter

Figure 2.4: Macroblock mode subdivisions. 16×16 and 4×4 for intra coding, 16×16, 16×8, 8×16 and 8×8 for inter coding. A single 8×8 block can be further partitioned into blocks of size 8×4, 4×8 and 4×4.

$$cost(\mu) = D(\mu) + \lambda R(\mu) \tag{2.4}$$

λ is a parameter that can be varied to favor either a small bitstream or low distortion.

The difference between the modes is the way a macroblock is partitioned into submacroblocks. The goal of the mode decision step is to find the most efficient representation. For example, a 16×16 macroblock can be divided into smaller macroblocks of size 8×8, each of which is coded separately with its own motion vector and transform coefficients. For example, when the picture changes texture inside the macroblock, splitting the macroblock into smaller macroblocks may yield a smaller representation than a single transform of the entire block. The encoder evaluates all modes and searches all possible motion vectors before finally selecting the mode with the lowest cost.

For intra-coded macroblocks (see Figure 2.4(a)), this step consists of checking the two possible macroblock subdivisions (one 16×16 block or 16 4×4 blocks). For the 16×16 mode 4 prediction directions are available, and in the 4×4 mode 9 prediction directions are searched. This prediction entails using the sample values of adjacent, previously coded blocks to approximate the sample values of the current block, and only coding the difference between the approximation and the actual value as described above. Intra prediction is similar to inter prediction, but heavily limited in scope. The mode with the lowest combined R-D cost is selected as the final mode for encoding.

For inter-coded macroblocks (see Figure 2.4(b)), the encoder tries a special SKIP mode to see if the entire macroblock can be predicted without storing any coefficients at all, does a motion search for every one of 7 possible macroblock partitions (16×16, 16×8, 8×16, 8×8, 8×4, 4×8 and 4×4) and selects the one with the lowest cost among the inter and intra modes. Motion search is explained further in the next section.

2.6 Motion Estimation

Motion estimation is done for every inter mode except SKIP (the SKIP mode is explained in Section 2.6.2). Motion estimation, or motion compensation as it is sometimes called, takes advantage of the fact that in a typical video the only change between two successive frames will be either camera movement, or movement of an object inside the scene. As a result, there will be duplicate content in the frames. If the encoder can exploit this redundancy by describing the movement between frames, a lot of space in the bitstream can be saved by because less information will have to be encoded.

This is achieved by searching for a position in the available reference frames for sample values having minimal distortion with respect to the current macroblock. The only

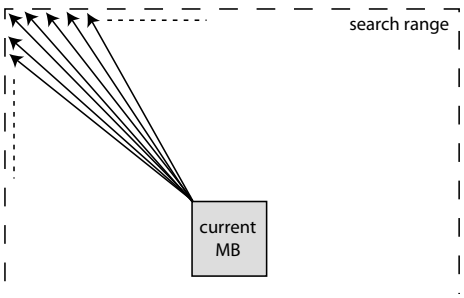


Figure 2.5: Full Search algorithm. All candidate motion vectors within the search range are tested, causing a calculation of the cost function for every vector.

thing that needs to be encoded then is the difference between the sample values of the source macroblock and the sample values in the reference frame at the position where the best match has been found. This difference is called the "residual".

Because both the residual and the motion vector have to be encoded, the best reference block is one that minimizes the cost function:

$$cost_{ME}(\vec{mv}) = D_{ME}(\vec{mv}) + \lambda R_{ME}(\vec{mv}) \tag{2.5}$$

Where $D_{ME}(\vec{mv})$ is the distortion obtained between the current macroblock and the macroblock in the reference frame displaced with the vector $\vec{mv}$, measured in SAD or SSD, and $R_{ME}(\vec{mv})$ is the number of bits required to encode the motion vector in the final bitstream. λ is used to weigh the domains of the two metrics, and is determined experimentally by the encoder developer.

The goal of this cost function is to compare candidate motion vectors. If a motion vector $\vec{a}$ has a lower cost than a motion vector $\vec{b}$, encoding the block with it will take up less space in the bitstream using $\vec{mv} = \vec{a}$ than using $\vec{mv} = \vec{b}$, and hence the R-D performance will be better.

To obtain the best compression, we have to search for the best motion vector $\vec{mv}$ using a motion estimation algorithm: iterate over a set of candidate motion vectors, calculate the value of the cost function for every one, and select the one with the lowest cost.

2.6.1 Motion Estimation Algorithms

The goal of the motion estimation algorithm is to find a relative motion vector that minimizes the cost function specified in the previous section.

Full Search

The simplest approach to this problem is to simply check all possible candidates motion vectors and select the best one (Figure 2.5). This algorithm is called Full Search, and it simply calculates the cost function of all (dx, dy) within some fixed range called the "search range". Full Search is a very inefficient algorithm, because it checks many

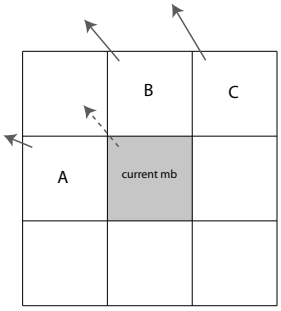


Figure 2.6: The Predicted Motion Vector (*PMV*) for the current MB is the median MV of the previously encoded macroblocks A, B and C.

vectors that have a low probability of being good candidates. To improve the speed of motion estimation, so-called "fast motion estimation algorithms" have been developed, which use heuristics to efficiently navigate through the search space.

These algorithms have the typical disadvantage of heuristic search that they can get stuck in a local instead of a global minimum. The increased efficiency is usually worth the trade-off of decreased accuracy however, and motion estimation algorithms are tested for robustness on a variety of testing sequences.

A good way of improving the efficiency of the motion search is to approximate a good initial motion vector for the search based on the motion detected in neighboring macroblocks, and then refining that vector with a heuristic search algorithm. In fact, this has even become a part of the H.264 standard, in the form of the Predicted Motion Vector.

2.6.2 Predicted Motion Vector

Without extra information, a motion search would have to start at the displacement vector $(0,0)$, and try to detect the motion in the scene from that.

However, in a real video sequence, macroblocks will typically move in conjunction if they are part of a larger moving object or if the camera is panning. This means that adjacent macroblocks typically have the same motion vector. The Predicted Motion Vector (*PMV*) takes advantage of this fact. The *PMV* is calculated based on previously processed adjacent blocks (Figure 2.6), and is used as the starting point of motion estimation. This can significantly speed up the motion estimation process [6].

Additionally, if the encoder decides that the *PMV* is "good enough", meaning that the distortion is below some threshold, it may decide to encode the current macroblock as a SKIP block. In essence, this is equal to encoding it with $MV = PMV$ and $D = 0$, but no bits have to be used in the bitstream to encode this information. At the decoder side, the referenced block is completely copied in place of the current block.

2.6.3 Uneven Multi-Hexagon Search

A heuristic motion search algorithm is used to refine the initial motion vector estimate. Uneven Multi-Hexagon Search (UMHS) [7] is the standard search algorithm used in

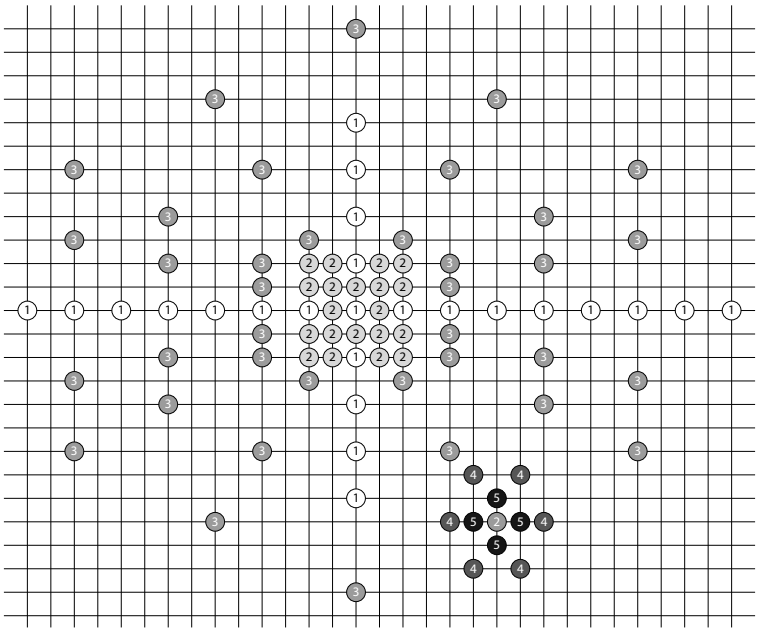


Figure 2.7: All stages of Uneven Multi-Hexagon Search. (1) asymmetrical cross search; (2) block search; (3) hexagon search; (4, 5) refined hexagon search.

H.264. The algorithm consists of 4 separate stages, shown graphically in Figure 2.7. After every stage, the best motion vector candidate (i.e., the one that has produced the lowest value for the cost function) discovered, is used as the starting vector for the next stage.

The stages used in UMHS are as follows:

1. First, an asymmetrical cross search is performed (points marked '1'), which covers an area twice as wide as it is high based on the observation that in most videos movement tends to be horizontal instead of vertical.

2. Then an exhaustive search of a 5×5 block is performed (marked with '2') around the step '1' vector that had the lowest cost. In the example figure, the center vector is chosen.

3. Then a multiple hexagon search is performed ('3') around the minimal cost vector from step '2'. In the example, again the center vector is chosen.

4. Finally the motion vector is refined using an iterative hexagon search (points marked '4' and '5').

In between each step, a check for early termination is done. If the distortion encountered so far is close to the minimal achieved distortion (SAD) among the blocks *A*, *B*, and *C* from Figure 2.6, the search skips directly to the final refinement of steps 4 and 5. The rationale here is that adjacent blocks are very likely to have similar distortion, so the search can be terminated early, thereby saving processing time. More information on the early termination mechanism of UMHS is available in [8].

2.6.4 Subpixel refinement

To improve the accuracy of motion estimation, H.264 allows motion vectors with a resolution of a quarter of a pixel. For every mode, after performing a fullpixel search with the UMHS algorithm described in the previous section, another step of subpixel refinement is done in two stages: first all 8 halfpixel vectors around the best fullpixel vector are checked, and then all 8 quarterpixel vectors around the best halfpixel vector are checked.

The halfpixel samples are obtained by interpolating the fullpixel values by taking the weighted average of 6 surrounding pixels, while quarterpixel samples are obtained by averaging fullpixel and halfpixel samples.

2.7 Summary

In this chapter, we have given an overview of video encoding for the H.264/AVC format. We have described the encoder architecture, and delved deeper into mode decisions and motion estimation, which are the most important aspects with regard to this thesis. The next chapter describes our approach to making the encoder complexity scalable.

Chapter 3

Complexity Scalable Video Encoding

In which we describe our approach to adapting the H.264 encoder so we can bound the complexity at the frame level and to minimizing the distortion incurred from reducing the complexity.

In this chapter, we describe our approach to adapting the H.264 encoder so that we are able bound the required complexity at the frame level while maximizing quality. The chapter is comprised of two parts. First, we introduce a technique for achieving complexity scalability at the macroblock level. Next, we introduce an allocation strategy that allocates the frame budget to macroblocks in such a way that distortion is minimized.

3.1 Complexity

Before we can define complexity scalability, we must first define complexity and how we measure it. Complexity is the time needed for a computer to execute a program and it can be measured in either processor instructions, clock cycles or seconds. All of these three units can be converted into one another, as there are fixed and well-understood mappings between them.

A program is a set of instructions for the computer's Central Processing Unit (CPU). The CPU executes the instructions of a program in discrete steps called "clock cycles". Every CPU has a "clock speed", which is the number of cycles it performs per second. These days, clock speeds are on the order of billions of cycles per second, or several GHz.

Every instruction of a program takes some fixed number of clock cycles to execute. How many instructions depends on the specific instruction and the hardware support provided by the processor. For example, an ADD instruction typically takes less clock cycles to execute than a MUL (multiply) instruction, although if there is dedicated hardware in the processor for multiplication, both may take only 1 clock cycle. We can

map from instructions to clock cycles by summing the clock cycles required for every instruction[1].

The number of clock cycles can be divided by the CPU's clock speed to obtain the actual "wall-clock" execution time of the program. The reverse mapping on the other hand, from wall-clock time to clock cycles, can be used to translate the deadline in seconds of a program in a real-time system into an upper bound of clock cycles that the program is allowed to spend executing without exceeding the deadline.

Our goal is to be able to arbitrarily bound the complexity of the video encoding process, meaning limiting how many instructions are executed so some clock cycle limit (and ultimately a time limit) is not exceeded. The actual bound can to be determined upon execution and depends on the CPU's architecture, clock speed, desired maximum power consumption and finally time-sharing load.

3.2 Scalability Mechanism

We need to identify an area of the encoder that we will modify to achieve complexity scalability. The requirement is that we must be able to reduce the amount of instructions that are executed, while still producing an output bitstream that is valid according to the H.264 format specification (i.e., can be decoded). This means that we cannot simply abort processing a frame when the complexity bound is reached. Instead, a part of the encoder must be selected that performs operations which are intended to improve encoder R-D performance, but are not *strictly* necessary for producing a valid output bitstream.

It is allowed, and even expected, that the distortion we achieve doing reduced computation will be higher than if we did full computation. Another way of putting this is that the quality will decrease. In the second part of this chapter, we will look for ways to minimize the introduced distortion.

3.2.1 Alternatives

Below we list some alternatives for reducing the computational need of the encoder.

Modifying the input picture stream's spatial and/or temporal resolution

This is done in [1]. By reducing the input stream's resolution we can reduce the number of macroblocks that have to be processed per time interval, thereby also reducing the complexity requirements of processing by some fixed factor F, which is the ratio of macroblocks per frame before and after resizing.

Due to various early termination heuristics used in the encoding algorithms, the complexity needs of the video stream are data dependent, and cannot be known a priori. Scaling the computational need by a factor F does not change the fact that we cannot

[1] This view is a bit simplified as CPU architectures these days are highly complex and try to employ caching, pipelining and parallelism to increase performance. The general principles still hold, though, and if all details are known a mapping can still be established.

predict the complexity needs of a frame beforehand. Furthermore, changing the resolution is not really part of the encoding process but rather of pre and post processing, and can be done in addition to the rest of the scheme described in this thesis. Therefore, we will not treat it further.

Restricting the number of Hadamard transforms

This approach is taken in [1], where the number of macroblocks that get transformed is bounded, and in [4], where the number of coefficients that are calculated in every transform is bounded. Since every transform computation consists of a fixed amount of mathematical operations, this approach makes it possible to place a hard bound on the number of computations that is performed.

Restricting the search for intra predictions

During the mode decisions for the intra coding modes, a number of prediction modes are checked, where samples from neighboring blocks can be copied from one of 4 directions for the 16×16 block or one of 9 directions for the 4×4 block, to predict the sample values of the current block. Every prediction check requires a calculation of the distortion for that choice. By restricting the number of prediction alternatives that are checked, the complexity requirements can be bounded. This technique is presented in [9] with an algorithm that minimizes distortion.

Restricting the search for inter predictions

When each of the 7 inter mode partitions other than the SKIP mode is checked for a macroblock, a number of motion searches have to be performed, one for every subblock in the partition. Every step of the motion search incurs a calculation of the distortion between the source macroblock and the reference macroblock at that point. We can bound the complexity of this process by limiting the number of distortion computations that are done, as is done in [1].

Conclusion

Not every part of the video encoding process has the same computational requirements. It is well-known that motion estimation is the most computationally intensive, taking up to 60–80% of total encoding time [7]. Furthermore, motion estimation is the step in encoding that introduces the most variance into computational requirements, depending on the contents of the video (unlike for example the transform computation or intra predictions). This means that if we do not target motion estimation for complexity scalability, we still cannot predict actual complexity.

Motion estimation is therefore both a required target for complexity scalability and also the target with the most potential impact on the entire complexity of the encoder. In this thesis, we focus our efforts on making the motion estimation process complexity scalable.

3.2.2 Complexity Scalable Motion Estimation

We want to be able to bound the complexity consumed by the motion estimation step of video encoding. Because all macroblocks in a video are analyzed and encoded separately, we focus on the motion estimation of a single macroblock.

Motion estimation consists of the examination of a number of different inter mode partitions, each of which consists of the motion search of a number of submacroblocks (see Section 2.5). Every step of motion search in turn consists of the computation of the distortion of a motion vector, plus some arithmetic to calculate the next vector candidate. The measure of distortion used is typically SAD (see Section 2.1).

The complexity cost of a SAD computation dominates the cost of a single motion search step and the cost of motion estimation in general. Thus, we can view the entire motion estimation process as a giant sequence of SAD computations. By reducing the number of SAD computations performed, we can save complexity. Of course, this will lead to fewer candidate motion vectors being examined, and the less motion vectors are examined, the less likely it will be that the optimal motion vector will be found. This is why reduced complexity will lead to reduced R-D performance.

Complexity Budget

To make the complexity of the motion estimation of a macroblock controllable, we introduce a "complexity budget". The complexity budget will determine the number of motion vectors the search algorithm is allowed to examine. Equivalently, it signifies how many SAD computations can be performed.

We choose this as the unit for complexity budget instead of a unit like clock cycles, as it is independent of the physical machine the encoder is running on. At the same time, it is a unit that is convenient to measure and to reason about in the context of the encoding process, and it is trivial to modify the motion estimation code such that the budget is observed: every extra vector examined decreases the budget with 1, until the budget is exhausted.

The actual complexity of a SAD calculation, in a unit that relates to the physical machine such as clock cycles, can be determined either through measurement or through analysis of the code. This knowledge can then be used to determine the maximum complexity budget for a given upper bound in clock cycles.

Changes to the Motion Estimation Algorithm

Since we have chosen a convenient unit of complexity, the changes to the motion estimation algorithm are trivial. On every inspection of a motion vector and the calculation of the SAD for that vector, the budget is decreased by one. When the budget is exhausted, the search is terminated, and the best motion vector found up until then will be the final motion vector for the macroblock.

It gets slightly more complicated when we consider that the budget for a single macroblock has to be spread over multiple mode partitions (16×16, 16×8, 8×8, etc...) and subblocks. There are two issues:

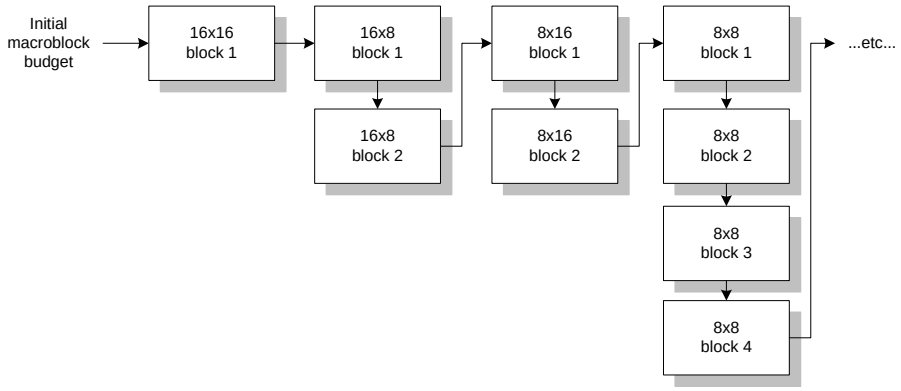


Figure 3.1: The order in which the motion estimation budget is spent on the (sub)macroblocks.

```
int sad(int x0, int y0, int w, int h, char** A, char** B) {
  int x, y;
  int sad = 0;

  for (y = y0; y < y0 + h; y++)
    for (x = x0; x < x0 + w; x++)
      sad += abs( B[y][x] - A[y][x] );

  return sad;
}
```

Table 3.1: Algorithm to compute the Sum of Absolute Differences between two blocks in two pixel arrays, in the C programming language. A and B are two-dimensional arrays representing the frames. The block to be compared has its top-left corner at (x_0, y_0) and is $w \times h$ pixels large.

1. Allocation of the budget to different partitions; and
2. The budget needs to be calibrated to some unit. We will select the evaluation of a 16×16 block as the unit for our complexity budget. However, the evaluation of a smaller block size will consume less (actual) complexity, and should therefore also consume less of the budget.

We solve the problem of allocating to different partitions in the following fashion: we process the partitions in-order, from large to small as shown in Figure 3.1. The reason for this is that the larger block sizes yield most of the R-D gain (up to 80% for the first 4 versus all 7 block partitions [10]), and should therefore be given preference. We treat the smaller block sizes as mode "refinements", used only when the complexity budget allows it, i.e., when the higher-mode motion searches have been fully completed.

As for the second point: the actual complexity cost of a motion vector evaluation mostly depends on the cost of the SAD computation, and the SAD cost of the SAD computation is linearly proportional to the number of pixels examined, as can be seen from the algorithm to compute SAD in Table 3.1. The complexity is proportional to $w \cdot h$, that is, the area of the block being examined.

Because the unit of our budget is the SAD computation of a single 16×16 block, smaller blocks consume only fraction F_μ of a single budget point for every SAD computation, where F_μ is their area relative to the surface of the 16×16 block:

$$F_\mu = \frac{w_\mu \cdot h_\mu}{16 \cdot 16} \tag{3.1}$$

Where w_μ and h_μ are the width and height of a submacroblock in mode μ.

With this scheme, we achieve a linear relationship between our own unit of budget (the number of 16×16 SAD computations) and actual consumed CPU complexity, as shown in our experimental result (see Section 4.1).

It should be noted that the UMHS algorithm [7] contains early-termination conditions [8], intended to abort the search when it is unlikely that a better match will be found. We have not changed these provisions, as they have been tested and accepted as a part of the standard. However, the presence of these early termination conditions makes it possible that not all of the allocated complexity budget is actually consumed. It is up to a control algorithm higher up in the allocation hierarchy to make sure this leftover budget is used effectively.

3.2.3 Complexity Model of a Frame

The mechanism introduced in the previous section controls the complexity of encoding a macroblock using a complexity budget, representing the number of 16×16 SAD computations that the encoder can do. If we know the actual complexity of one such SAD computation on our machine, and the complexity of the rest of the non-scalable operations, we can build a model that we can use to determine the actual complexity of encoding a video frame, based on our complexity budget:

$$C_{frame} = \alpha \cdot B_{frame} + \beta \cdot M + \gamma \tag{3.2}$$

$$B_{frame} = \sum_{0 \leq m < M} B_m \tag{3.3}$$

Where M is the number of macroblocks, B_m is the complexity budget for macroblock m and B_{frame} is the complexity budget for the entire frame. C_{frame} is the complexity required for encoding the entire frame in clock cycles. α is the number of clock cycles required for performing a single 16×16 SAD computation, β is the number of clock cycles required for additional non-scalable processing of a macroblock (which includes intra mode decision, quantization and transform), and γ is the overhead cost in clock cycles of processing the frame.

Once the values of α, β and γ have been determined for a particular processor, this model can be used to find the complexity budget B_{frame} that has to be used to bound the frame's complexity to obtain a desired actual complexity C_{frame}.

Note that in a fully intra-coded frame (an I-frame), inter modes are not allowed and so are not evaluated. This means that $B_{frame} = 0$, and that I-frames always require a fixed

amount of complexity that can be calculated as $C_{frame} = \beta \cdot M + \gamma$. This value is the minimum of the scalable encoder's complexity range.

This thesis does not deal with the question of how C_{frame} should be chosen. This entirely depends on the environment in which the scalability mechanism is used. For example, C_{frame} can be chosen to be some constant. If due to early termination during motion estimation the consumed complexity budget is smaller than the assigned complexity budget, the system can choose to add this leftover budget to the budget of the next frame (if buffer space allows), or not, depending on whether the reason for using complexity scalability is to meet encoding deadlines or simply to conserve power.

3.3 Frame Budget Allocation

We now have the ability to encode macroblocks in a complexity scalable way and we are able to bound the complexity of the frame by selecting appropriate complexity budgets for every macroblock (as shown in (3.2) and (3.3)).

However, when we want to bound the frame complexity to a given actual complexity C_{frame}, this tells us what the maximum value of B_{frame} can be, but not how to choose the B_m. We can use any allocation we choose, but the way we allocate the complexity will have an effect on the resultant R-D performance of the output video. Our goal is to minimize:

$$\min D(B_0, B_1, \ldots, B_{M-1}) \quad \text{s.t.} \quad \sum_{0 \leq m < M} B_m \leq B_{frame} \tag{3.4}$$

In this section, we will present an allocation algorithm that improves distortion over the default uniform allocation and that performs equal to better than the algorithm presented in [1].

3.3.1 Constraints

Every extra point of budget that we allocate to a macroblock means at least one other motion search step is performed, having the potential to improve the best motion estimation vector match for that macroblock and thereby improving the R-D performance of the encoder. However, this improvement is not guaranteed, and we also cannot predict at what point the improvement will occur (see Figure 3.2). In fact, the whole reason that we perform motion search is exactly because we do not know the position of the best motion vector beforehand. We have to perform the actual motion search of all macroblocks before we can know the shape of their C-R-D curves, showing how the R-D cost changes as a function of complexity (C).

We can also not pick and choose, in the middle of encoding a frame, which macroblock to do an extra step of motion estimation on. Though macroblocks of a frame are encoded separately, they are not independent: the predicted motion vector of a macroblock depends on the resultant motion vector of the blocks preceding it. This means that the blocks must always be encoded in the order specified by the standard, which is scan order.

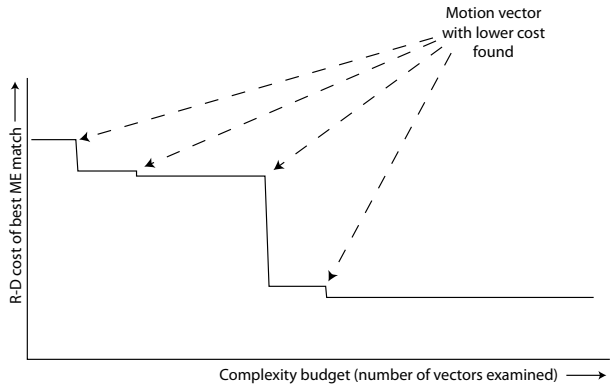


Figure 3.2: Example of the Complexity-Rate-Distortion (C-R-D) curve of a macroblock as the complexity is increased. The R-D cost of a macroblock is a monotonically non-decreasing function of the assigned complexity (C) whose shape cannot be predicted.

Since we do not have random access to macroblocks, the allocation of complexity must be done before the first macroblock is processed, and therefore without any information about the shape of the C-R-D curve of the macroblocks at all.

We can never know the optimal allocation before encoding a frame, so we will have to estimate it from other information that is available to us. This means that will have to rely on statistics that are gathered in previous frames to predict the complexity budget needs of the macroblocks in the current frame.

3.3.2 Allocation Considerations

The simplest allocation algorithm would be to distribute the available complexity budget equally over all macroblocks ($B_m = \frac{1}{M} \cdot B_{frame}$). However, this allocation is probably suboptimal. Increasing a macroblock's budget increases the amount of motion estimation that is done for that block and increases the chances of a good motion vector match, but only if such a better match is available.

- It can be that the predicted motion vector is already optimal (in case of constant movement through a large part of the video), in which case adding extra complexity will not improve the match.
- It can be that no good match is available because too much has changed in the scene, in which case the block will probably be intra-coded and all effort spent on motion estimation is wasted.

Because we can not inspect the situation for the macroblocks in the current frame, but instead have to rely on knowledge obtained in previous frames, we need some sort of metric and allocation algorithm to predict the complexity needs of a macroblock.

In [1], a method called Motion History Matrix (MHM) is introduced that allocates complexity to macroblocks inversely proportional to the amount of stillness detected in the macroblock. The idea here is that macroblocks containing a lot of movement will

need more complexity, while in areas of little movement the predictors will usually be sufficient and less motion search is needed. This work was implemented in H.263.

Our testing has shown that in H.264, MHM allocation does not yield significantly better results than the naive uniform allocation, and in sequences with a lot of movement or a shaky camera reduces to being the same allocation[2]. The addition of the Predicted Motion Vector to the H.264 standard means that the presence of motion by itself is not enough to predict complexity needs (as motion that can be predicted by the PMV does not need additional motion search).

We will therefore try to find an allocation algorithm that performs better. In our case, we base the allocation not on the movement in the video, but on the achieved distortion; the idea being that areas with high distortion need more work to improve than areas that already have low distortion.

In the rest of this section, we introduce our allocation strategy and algorithms. In the next chapter, we then compare our algorithms to the naive and MHM allocation to see which one gives the best performance.

3.3.3 Hypothesis

We hypothesize that a high distortion indicates a need and a possibility for improvement: if a macroblock had a high distortion in the previous frame, this is an opportunity to invest more computation into it in this frame to find better motion estimation match and decrease the achieved distortion.

On the other hand, the inverse could equally well be true: by investing complexity into blocks that have good quality, we prevent those blocks from going bad. Bad blocks may be unsalvageable anyway, leading to intra coding and wasted complexity.

Although our intuition says the initial hypothesis is more likely, we will try both approaches and see which one yields the best results.

3.3.4 Allocation Algorithms

In this section, we explain all allocation algorithms that we will evaluate to see which one yields the lowest distortion at any given complexity bound. Two of them will be reference algorithms and three of them will be new, each having two variants.

All algorithms follow the same pattern: the algorithms collect some metrics $x_{0,0},\ldots,x_{M-1,N-1}$ for every one of M macroblocks in N frames. A share of the frame budget is then allocated to a macroblock proportionally or inversely proportional to the share of its metric value to the sum of all metric values in a frame. In fact, the metrics represent weights.

The proportional allocation of the complexity budget $B_{m,n}$ to macroblock m in frame n as a function of the metrics $x_{0,n}\ldots x_{M-1,n}$ is calculated as follows (for this and all equations in this section $0 \leq m < M, 0 \leq n < N$):

$$B^{\rho}_{m,n} = \frac{x_{m,n}}{\sum_{0 \leq i < M} x_{i,n}} \cdot B_{frame}(n) \tag{3.5}$$

[2]For example, refer to the graphs of the sequences “Foreman”, “City”, “Crew” and “Bus” at 500 kbps in Appendix A.4.

Where $B_{frame}(n)$ is the frame complexity budget for frame n. Inversely proportional allocation is calculated as follows:

$$B^{\iota}_{m,n} = \frac{1}{M-1}\left(1 - \frac{x_{m,n}}{\sum\limits_{0\leq i<M} x_{i,n}}\right) \cdot B_{frame}(n), \quad M > 1 \tag{3.6}$$

With the restriction that $\sum_{0\leq m<M} x_{m,n} \neq 0$.

Re-use of Unused Budget

Because the motion search of a macroblock can terminate early when no further improvement of the motion vector is expected (see Section 2.6.3), it can be possible for the motion estimation to not fully consume the complexity budget allocated to it. To prevent this unused budget from being wasted, we re-allocate it to remaining macroblocks using the same proportional or inversely proportional allocation method. This is achieved by evaluating the allocation (3.5) and (3.6) not before encoding the frame, but before encoding each macroblock, using the proportion of the macroblock's metric to the sum of the *remaining* macroblock's metrics, and the *remaining* frame budget.

We define $B_{frame}(n,m)$ to be the remaining frame budget available before encoding macroblock m in frame n, calculated as follows:

$$B_{frame}(n,m) = B_{frame}(n) - \sum_{0\leq i<m} \overline{B}_{i,n} \tag{3.7}$$

Where $\overline{B}_{i,n}$ is the actual complexity budget consumed while encoding macroblock i in frame n. Now we can define proportional allocation with re-use as follows:

$$B^{\rho}_{m,n} = \frac{x_{m,n}}{\sum\limits_{m\leq i<M} x_{i,n}} \cdot B_{frame}(n,m) \tag{3.8}$$

Inversely proportional allocation with re-use is calculated as fllows:

$$B^{\iota}_{m,n} = \begin{cases} B_{frame}(n,m) & \text{if } M-m=1 \\ \dfrac{1}{M-m-1}\left(1 - \dfrac{x_{m,n}}{\sum\limits_{m\leq i<M} x_{i,n}}\right) \cdot B_{frame}(n,m) & \text{if } M-m>1 \end{cases} \tag{3.9}$$

Allocation with re-use will never allocate less budget to a macroblock than allocation without re-use, but it may allocate more if for some macroblocks $\overline{B}_m < B_{m,n}$. Macroblocks that have already been processed before the extra budget becomes available will not benefit from the extra complexity budget, only macroblocks that still have to be encoded.

The following sections will describe the metrics $x_{m,n}$ that are used in the various algorithms.

Uniform Allocation

Uniform allocation is the naive allocation strategy, which we only use for comparison purposes. Uniform allocation allocates the same amount of budget to every macroblock by giving every macroblock the same weight:

$$x_{m,n} = 1 \tag{3.10}$$

There is no difference between proportional and inversely proportional allocation with this metric.

Motion History Matrix (MHM)

The MHM allocation algorithm is presented in [1]. It calculates a measure of how "static" a block is (meaning a lack of movement is detected) and allocates less complexity to blocks that are deemed static. To this end, a Motion History Matrix is maintained, which contains a counter of the number of frames in a row in which the macroblock has not moved. "Not moved" is defined by the motion vector being equal to $(0,0)$.

The elements of the MHM are our metrics $x_{m,n}$. Initially, all values are set to 1, and after motion estimation of a macroblock m, the values are updated based on the result of motion vector $\vec{mv}_{m,n}$:

$$x_{m,n+1} = \begin{cases} x_{m,n} + 1 & \text{if } \mid \vec{mv}_{m,n} \mid = 0 \\ 1 & \text{if } \mid \vec{mv}_{m,n} \mid \neq 0 \end{cases} \tag{3.11}$$

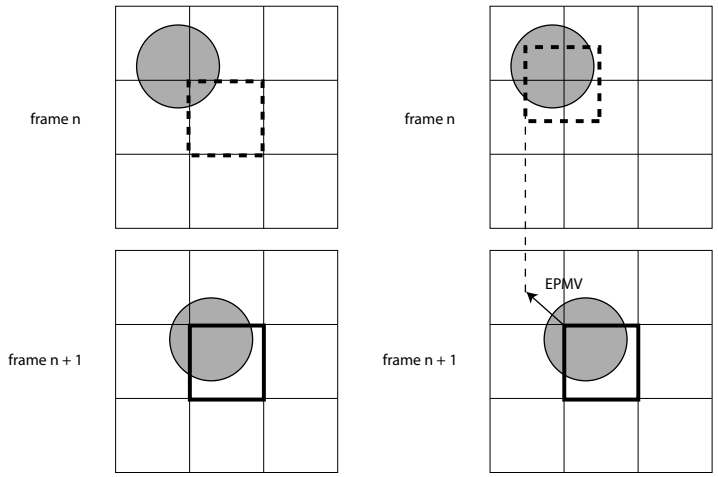
The allocation is done with the inverse proportional allocation strategy. This has the effect of reducing the complexity allocation to blocks that are mostly stationary.

The goal of this algorithm is to identify movement. Areas with movement will need more steps of motion search to find a good match than areas with low movement and so should get a higher budget. On the other hand, a weakness in this algorithm is that in scenes with a lot of movement, most of the motion vectors $\mid \vec{mv}_{m,n} \mid \neq 0$, which leads to $x_{m,n+1}$ constantly being reset to $x_{m,n+1} = 1$. In the limit, this reduces MHM to Uniform Allocation for high-motion sequences.

Distortion-based allocation

Our hypothesis is that the distortion achieved while encoding the macroblock in the previous frame is a better indicator of the need for complexity budget. Distortion is a measure independent of movement which more accurately reflects how "hard" it was to encode the macroblock. Consider a large object moving at a constant speed throughout the video: although all macroblocks comprising the object will be moving (thereby resetting their MHM value and ensuring they get a larger share of the complexity budget), their predicted motion vectors will already be the best motion vector available, and they will not benefit from additional motion search budget. Incidentally, the distortion will be low in this case.

We consider two possible measures of distortion, *Direct Distortion* and *Displaced Distortion* (see Figure 3.3). The first one is the distortion achieved encoding the current



(a) Direct distortion (b) Displaced distortion

Figure 3.3: Measuring distortion of a macroblock in the previous frame. The macroblock with the thick black border is the one currently being encoded. The distortion achieved encoding the square with the thick dashed border is used as the predictor for the distortion of the current block. 3.3(a): the same macroblock in the previous frame is used. 3.3(b): the macroblock position is displaced with the current macroblock's (estimated) predicted motion vector (EPMV).

macroblock in the previous frame (Figure 3.3(a)). The second one also calculates the distortion achieved encoding the block in the previous frame, but it will try and displace the position of the block with the motion vector of the current macroblock. This yields a better approximation of the distortion, by measuring the block of pixels in the previous frame that will be in the macroblock's location in the current frame.

We can not *actually* displace with the motion vector of the current macroblock however, because we can not know the real motion vector of the macroblock until after the motion estimation has actually been done. We therefore need an estimation of the macroblock's motion vector beforehand. A natural candidate would be the macroblock's Predicted Motion Vector (PMV). Unfortunately, a macroblock's PMV depends on neighboring macroblocks in the same frame, and since we have to do complexity allocation before any encoding is done, we cannot use the real PMV. We therefore have to approximate the PMV. We will resort to using the block's motion vector in the previous frame. This is an approximation that requires that movement is constant between frames.

As our measure of distortion, we will use SAD. The metric for direct distortion in frame $n+1$ is:

$$x_{m,n+1} = SAD(\ F_n(\vec{p_m})\ ,\ F'_n(\vec{p_m})\ ) \tag{3.12}$$

And the metric for displaced distortion is:

$$x_{m,n+1} = SAD(\ F_n(\vec{p_m} + \vec{mv}_{m,n})\ ,\ F'_n(\vec{p_m} + \vec{mv}_{m,n})\ ) \tag{3.13}$$

Where F_n is source frame n and F'_n is reference frame n (frame n after being encoded and decoded). $F_n(\vec{v})$ is the 16×16 block in frame n at the position specified by vector $\vec{v}$, $\vec{p_m}$ is the vector specifying the position of macroblock m and $\vec{mv}_{m,n}$ is the motion vector of macroblock m in frame n.

We will evaluate both the proportional and inversely proportional allocations of both of these metrics.

Residual-based allocation

Although we only control the motion estimation process of macroblock encoding, the metrics gathered by the distortion-based allocation algorithm of the previous section are influenced not only by the motion estimation performed, but also by the quantization imposed by the later stages of encoding. If there were no or very little quantization, all distortion values would all be very low because the motion estimation residual could be encoded without loss. If all distortion values are very close to each other, the allocation again reduces to Uniform Allocation, with all macroblocks having the same weight. This is why we also measure a distortion that is only dependent on the motion estimation process: the residual.

The residual is the difference between the sample values of the macroblock to be encoded and the final motion estimation match (see Section 2.6). It is also a measure for how hard it is for the encoder to encode the macroblock (as a bigger residual means that more values have to be compressed and stored in the bitstream), specifically, how hard it is to find a good motion estimation match for the macroblock. The same assumption holds as with the distortion-based allocation: a larger residual indicates bad encoder performance. By allocating additional complexity budget to macroblocks with a large residual, we expect to improve the motion estimation match, thereby decreasing the size of the residual and increasing the encoder's performance.

We measure the residual also in SAD. The metric for macroblock m in frame $n+1$ is defined as follows:

$$x_{m,n+1} = SAD(\ F_n(\vec{p}_m)\ ,\ F'_{n-1}(\vec{p}_m + \vec{mv}_{m,n})\ ) \tag{3.14}$$

3.4 Summary

Figure 3.4 shows an overview of the metrics used for budget allocation in each of our proposed allocation algorithms. In the next chapter, we will present our experimental results and compare our proposed algorithms to the Uniform Allocation and MHM allocation.

Figure 3.5 shows the allocation hierarchy for the entire system, from frame complexity down to fullpixel motion search. There are a number of allocation levels in the hierarchy that our research did not touch on, which we recommend as future work. Notably, an allocation between intra and inter mode decisions, and an allocation between fullpixel and subpixel search can yield improved scalability.

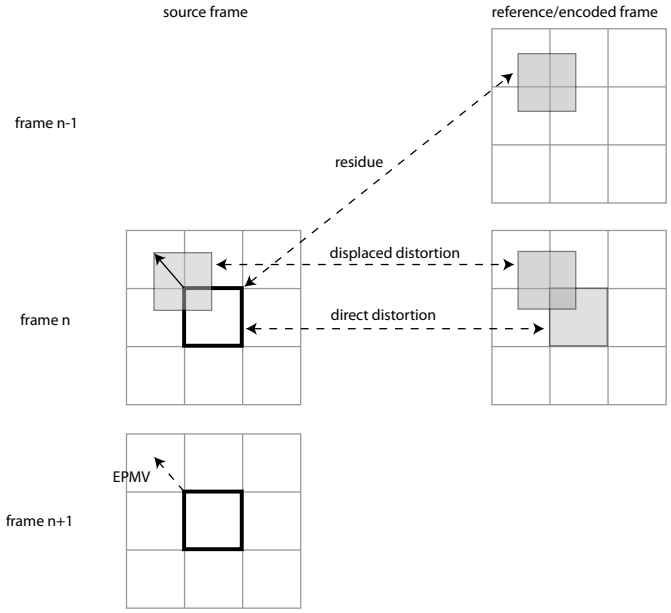


Figure 3.4: Schematic overview of the three distortion metrics used by our proposed allocation algorithms. Top to bottom in this diagram shows the progress of frames through time; we are currently encoding frame $n+1$. The frames on the left are source frames, while the frames on the right are the encoded versions of every frame. The squares indicate the area that is used for each distortion computation. The solid arrow is the MV of the macroblock in frame n, which is used as EPMV in the current frame.

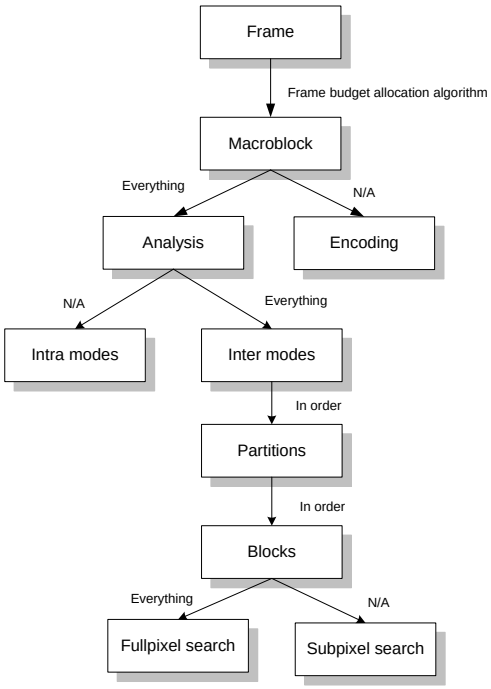


Figure 3.5: Overview of the allocation hierarchy from frame budget down to fullpixel motion search. The boxes represent physical video elements or logical video encoding steps, while the text next to the arrows indicates how an element's complexity budget is allocated to its constituents. Arrows to steps that are not currently complexity scalable in our mechanism are labeled "N/A" (not applicable).

Chapter 4

Experimental Results

In which we evaluate the scalability mechanism introduced in the previous chapter for predictive power, and the allocation algorithms for qualitative performance.

This chapter presents our findings evaluating the complexity scalability mechanism and allocation algorithms introduced in the previous chapter. We show that the complexity scalability mechanism allows us to bound complexity to an arbitrary number of clock cycles. We also show that our budget allocation algorithms yield lower distortion than either MHM or Uniform Allocation using rate-controlled encoding.

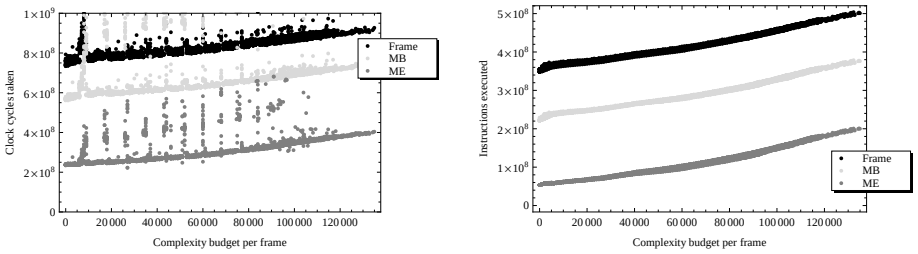
4.1 Complexity Scalability Mechanism

We have implemented the complexity scalable motion estimation mechanism outlined in Section 3.2 in the H.264/AVC reference encoder software, JM 14.2. Our mechanism introduces a complexity budget, limiting the number of motion vectors that are examined during motion estimation. The unit of our budget is the examination of a single vector for a 16×16 block of pixels, and each vector examination consumed a part of the budget proportional to the size of the examined block.

The goal of introducing the motion estimation complexity budget is to bound the effectively consumed computational complexity of encoding a single frame, measured in some physical measurement like clock cycles. This goal is satisfied if there must be a linear relationship between consumed points of the complexity budget and the actual consumed complexity. In this section, we will evaluate whether our proposed mechanism meets this goal.

To evaluate whether there is a linear mapping from complexity budget to actual complexity measured in clock cycles, we have encoded a short fragment of a testing video sequence at various complexity bounds[1]. The video is encoded without subpixel motion estimation, and with a search range of 32 pixels. During every encode, the complexity bounds are constant and equal for every frame. The result is that every frame is encoded at every complexity bound.

[1] We have encoded the "Foreman" video sequence at CIF resolution (352×288), for 30 frames. The mechanism is independent of the video, so no further videos have been examined.



(a) Clock cycles

(b) Instructions

Figure 4.1: Actual complexity of encoding, measured in clock cycles (4.1(a)) and CPU instructions (4.1(b)), as a function of the complexity budget B_{frame}. The top point cloud ("Frame") is the total complexity spent encoding a frame, the middle point cloud ("MB") is the total complexity spent on macroblock analysis, and the bottom point cloud ("ME") is the complexity spent on motion estimation.

For every frame, we measure the *consumed complexity budget*, which can be slightly less than the initially allocated budget due to rounding errors and failure to re-use, and the *actual consumed complexity* in clock cycles. This yields a set of points that relate complexity budget (B_{frame}) to actual complexity (C_{frame}). If there is a linear relationship between complexity budget and actual complexity, all points should lie on a straight line.

We've plotted the set of points in a graph (see Figure 4.1(a)). To gain a greater insight into the complexity consumption of the encoder, we've measured three different complexity consumptions:

- Frame: the total computational complexity spent encoding the frame.
- Macroblock (MB): the sum of all time spent analyzing and encoding the macroblocks the frame. The difference between this and the previous value indicates the overhead processing time of a frame.
- Motion estimation (ME): the total computational complexity spent on motion estimation during the frame. The difference between this value and the previous one indicates the amount of non-motion estimation processing done for a macroblock.

The data of Figure 4.1(a), collected on an Intel Pentium 4 machine with a clock speed of 3.0 GHz running Linux, is rather noisy however. This is due to the fact that we've run our experiments on a time-sharing system. The CPU has a Timestamp Counter (TSC), which tells us the number of clock cycles elapsed since the CPU powered on. We measure the computational time spent by taking the delta between two measurements of the TSC over some piece of code.

However, on a time-sharing system, our encoder process can be pre-empted and some other process can be run. In the meantime, the TSC will keep on increasing and the delta will no longer reflect the running time of just our application but include the cycles spent executing the pre-empting process.

To prove that the noise of Figure 4.1(a) is due to pre-emption, and not due to the encoder not obeying the complexity budget, we need a different measurement.

Most processors offer another way to profile applications, with a feature built into the CPU called Performance Counters. With a driver that integrates into the operating system [11], we can access various metrics regarding the CPU performance's. The Intel *Pentium* performance counters do not allow us to measure the clock cycles spent per process. However, they do let us measure the number of machine code instructions executed per process.

Figure 4.1(b) shows the complexity consumption of the video encoder, measured in executed instructions. The noise from Figure 4.1(a) is absent. The measurements for both graphs have been collected during the same run of the encoder, which means the exact same code is executing at the exact same complexity bound. Because the exact same set of instructions has been executed to obtain both graphs, we can conclude that the noise in Figure 4.1(a) is solely due to the effects of preemption. Ignoring the noise, our mechanism establishes a linear relationship between complexity budget B_{frame} and actual complexity C_{frame}.

Looking at the measurements of the clock cycles, every one of the three point clouds approximates a line. Knowing that there are 396 macroblocks in a video of CIF resolution ($M = 396$), we can estimate the parameters of the model for the frame complexity of equation 3.2:

$$C_{frame} = \alpha \cdot B_{frame} + \beta \cdot M + \gamma$$

$\alpha =$ the slope of the line approximated by the ME point cloud

$$\approx \frac{3.4 \cdot 10^8 - 2.4 \cdot 10^8}{1 \cdot 10^5} = 1000$$

$\beta =$ the value of the MB line at $B_{frame} = 0$ over M

$$\approx \frac{5.75 \cdot 10^8}{396} = 1.45 \cdot 10^6$$

$\gamma =$ the difference between the Frame line and the MB line

$$\approx 7.5 \cdot 10^8 - 5.5 \cdot 10^8 = 2 \cdot 10^8$$

Note that these values for α, β and γ are only valid for the machine the experiments were performed on. However, on that machine they are the same for every video sequence (only varying in M), so after an initial calibration to obtain the right values the model can be used on any machine.

We can now use this model to predict the complexity requirements of any frame at any resolution bounded to a particular budget, or alternatively, find the budget necessary to bound the complexity to the desired level. This means our goal of limiting the complexity requirements of encoding a frame to an arbitrary point (within the range allowed by the encoder, of course) is achieved. Furthermore, the encoder can operate along the entire complexity range, by selecting the appropriate complexity bound.

What we can tell from the measurements, is that the encoder has a large amount of overhead for processing a frame, and that it does a large amount of non-scalable work related to motion estimation. The scalability afforded by the scalable motion estimation

is actually not very good (about 16%). It should be noted that JM is a piece of software that is designed for research purposes, and favors code clarity over performance. For example, it does a lot of memory block copying, which is not conducive to performance. In an optimized encoder, the difference would be a lot larger, making motion estimation indeed the most expensive task and making the complexity scalability a lot more effective.

In fact, our experiments with x264, an open source H.264 encoder which has been heavily optimized for performance, show that the complexity scalability implemented in that codec offers up to 60% scalability.

4.2 Complexity Allocation Algorithms

We've shown that it is possible to select a complexity budget for the frame B_{frame} such that a particular complexity bound measured in clock cycles is satisfied. In this section, we evaluate the allocation algorithms specified in Section 3.3, whose purpose it is to allocate the selected complexity budget to macroblocks in such a way that R-D performance is maximized.

We evaluate these algorithms with respect to the achieved joint C-R-D performance. To evaluate the C-R-D performance, we fix the bitrate of the encoding using the standard rate control algorithms of the JM encoder, and evaluate the distortion as a function of complexity budget.

However, we will not present the distortion numbers directly. Instead, we will present our results in the PSNR scale, as that is the traditional scale for presenting results with respect to signal fidelity. Our hope is that use of the PSNR scale will give the reader a better intuition of the quality levels involved in the results. To give an idea of the scale, 20 dB is not very good quality, 30 dB which is acceptable quality and 40 dB and higher are very good quality.

We have examined a number of testing video sequences at a fixed bitrate. The sequences are different video scenes captured from analog media and saved in an uncompressed format, commonly used to test the performance of video encoders. All are available on-line [12]. The sequences we used have been chosen to have medium to very high motion to showcase the differences between the allocation algorithms. The encoding bitrate has been chosen to yield a PSNR of around 30 dB, which yields reasonable quality without saturating the rate requirements of the video. The "Bus" sequence has also been encoded at a much lower and much higher bitrate, to see the effects of different bitrates (and hence, different quality ranges) on the algorithms. All videos were encoded without subpixel motion estimation, a search range of 32 pixels and 1 reference frame. See Table 4.1 for a full list of the sequences, encoding rates, and results achieved without complexity constraints.

For our comparative graphs, all videos were encoded with the same complexity budget for every frame. No re-use of leftover budget between frames was done. We measure quality in PSNR for every video as a function of the frame budget. Note that on the x-axis, we do not plot *consumed* budget, but *assigned* budget. We want the best possible quality for our available budget, so algorithms that achieve lower quality because they underallocate need to be penalized.

Sequence	Frames	Bitrate	Classification	Avg complex.	Quality
Foreman	300	250	Medium motion	$1.9 \cdot 10^5$	30.84 dB
City	600	500	Medium motion	$1.7 \cdot 10^5$	31.20 dB
Crew	600	250	Medium motion	$2.8 \cdot 10^5$	31.00 dB
Stefan	300	1500	High motion	$2.1 \cdot 10^5$	31.06 dB
Husky	250	4000	High motion, fine detail	$3.6 \cdot 10^5$	28.20 dB
Bus	150	500	Very high motion	$3.5 \cdot 10^5$	26.57 dB
Bus	150	1500	Very high motion	$2.5 \cdot 10^5$	31.64 dB
Bus	150	6000	Very high motion	$1.9 \cdot 10^5$	38.46 dB

Table 4.1: The list of testing sequences used and the rates tested. Each video is encoded at 30 frames per second. All videos are CIF resolution (352×288) with the exception of "Stefan", which is SIF resolution (352×240). Bitrate is in kbps, Quality is in dB PSNR. Average complexity is the average unbounded complexity budget consumption of each frame in the sequence, while Quality is the quality achieved at the given rate without a complexity bound.

Graphs of the full results are available in Appendix A. However, since there are so many, we will only show a couple of the measurement graphs in the following sections in the interest of brevity. The sequence that we will most frequently refer to is the sequence called "Bus", as it is the one that shows the most visible difference in algorithms during our testing[2].

Two conclusions can immediately be drawn from the graphs:

The results are noisy

There is a lot of noise in the graphs, especially the "Stefan" sequence and "Bus" sequence at 6000 kbps.

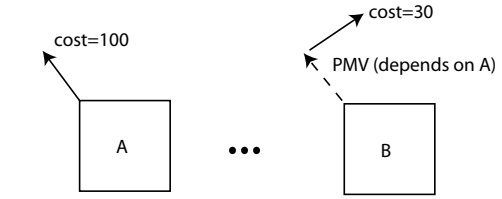
Apparently, it is possible that encoding with a higher complexity budget yields lower quality, which is the opposite of what is desired and expected. How can this happen?

One of the factors is the fact that macroblocks are highly dependent on each other. When the motion estimation for one macroblock yields a motion vector, that vector will influence the starting point motion vector of at least 3 of the following blocks. One extra point of budget, equal to one extra step of motion estimation, may lead to a better motion vector for block A, while at the same time changing the predicted motion vector for blocks B, C and D. If this leads to B, C and D missing their optimal motion vector, the global end quality is actually worse with the extra point of complexity than without. See Figure 4.2 for a graphical example.

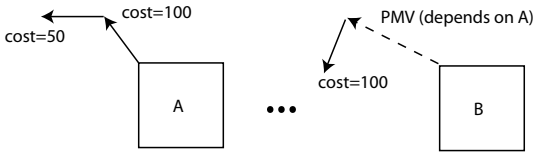
Another reason is that two consecutive points in the C-D graphs are actually from two different encodes of the video at two different complexity bounds. Since the complexity budget is different, this will affect the motion vectors and distortion of the macroblocks, which will change the distribution with respect to the previous point. This may ultimately lead to a lower overall quality than was achieved with a lower budget.

Especially in the "Stefan" sequence, the noise is very prominently visible. This is mostly due to the last few seconds of the video, where camera pans very rapidly over

[2] In the Bus sequence the camera constantly zooms out, which means that predictors are not as useful as when the camera is making a constant panning motion.



(a) Some budget allocated to block A



(b) One extra point of budget to block A

Figure 4.2: Visualizing the dependency between macroblocks. The PMV of block B is based on the MV of block A. In 4.2(a), both blocks have some complexity budget, and together the blocks achieve an R-D cost of 130. In 4.2(b), block A has an additional point of complexity budget, which changes its MV. Consequently, this changes B's PMV, causing its search algorithm to have a different starting point and ending up with a different final MV. Although the complexity budget has increased and A was able to decrease its cost, the total cost of both blocks has now increased to 150 instead of decreased.

a tribune filled with spectators. This part of the sequence contains a lot of fine details, which leads to a large difference between "hit" and "missed" motion vectors, and introduces a lot of noise. This noise is so large that it is even visible in the global video results.

Additionally, at higher qualities, the noise becomes more prominent in a PSNR graph. PSNR is based on the Sum of Squared Differences (SSD), but is inverted. As can be seen in Figure 4.3, as distortion decreases, the slope of the PSNR function increases, and the same difference in distortion will lead to a larger difference in PSNR. Therefore, at high PSNR values, any noise in the SSD values will become exaggerated on the PSNR scale. This is very visible in the "Bus" sequence at 6000 kbps.

Ignoring the noise introduced by these factors, we can conclude that the quality increases as a function of the complexity.

The effective scalability range is limited

There is only a small interval showing a difference between the algorithms. After some complexity threshold the quality of the video is saturated, and no or very little extra quality is gained from added complexity.

To quantify this, we have calculated at what point across the scalability range, meaning from minimum to maximum consumable complexity budget, 95% of achievable quality is obtained using Uniform Allocation. Minimum complexity is achieved when $B_{frame} = 0$; maximum complexity is the amount of budget consumed when we don't bound complexity. Because every frame can have a different maximum complexity,

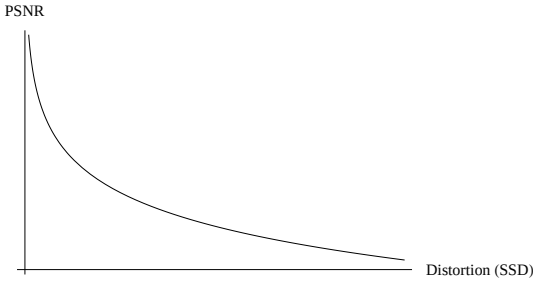


Figure 4.3: The function that calculates PSNR from SSD is of the form $PSNR = log(\frac{1}{SSD})$ (see Section 2.1). The shape of this function is such that the slope is higher the lower the SSD values get. I.e., as the distortion decreases, quality increases, and the variance between two values becomes greater.

Sequence	**Bitrate**	**Max cplx.**	**Cplx. at 95%**	**Fraction**
Foreman	250	$1.9 \cdot 10^5$	$2.7 \cdot 10^4$	0.14
City	500	$1.7 \cdot 10^5$	$2.7 \cdot 10^3$	0.01
Crew	250	$2.8 \cdot 10^5$	$2.9 \cdot 10^4$	0.10
Stefan	1500	$2.1 \cdot 10^5$	$6.2 \cdot 10^3$	0.03
Husky	4000	$3.6 \cdot 10^5$	$2.3 \cdot 10^4$	0.06
Bus	500	$3.5 \cdot 10^5$	$2.9 \cdot 10^4$	0.08
Bus	1500	$2.5 \cdot 10^5$	$1.1 \cdot 10^4$	0.04
Bus	6000	$1.9 \cdot 10^5$	$4.8 \cdot 10^3$	0.02

Table 4.2: At what fraction of the complexity scalability range of every video is 95% of the attainable quality achieved. The fraction of quality is measured in Sum of Squared Differences, since SSD is a linear scale as opposed to PSNR. "Maximum complexity" is defined as the averaged unbounded complexity of the frames in the sequence.

we aggregate all the maximums to a single value by taking the average. To quantify quality, we resort back to distortion as given by SSD. This is because SSD is actually a linear scale, in contrast to PSNR which is a logarithmic scale. We have measured the point at which 95% of the decrease in SSD is achieved between minimum and maximum complexity.

Table 4.2 shows the results. It can be observed that the saturation happens at a fraction of the budget of what the unbounded motion estimation process would normally consume, the largest fraction in this table being at 14%. This means that complexity can be reduced by a lot without adversely affecting quality, regardless of allocation algorithm. However, because it cannot be known beforehand at what complexity bound saturation occurs, it is still advisable to use the allocation algorithm with optimal performance in all cases.

Since there is hardly any discernible difference between the allocation algorithms after quality is saturated, in the next sections we will highlight only the areas where the complexity budget is limited and a difference is visible.

4.2.1 Global Measurements versus Per-Frame Measurements

It should be noted that the effectiveness of using different allocation algorithms varies heavily per frame. In some frames, all algorithms perform nearly identically, whereas in other frames the difference between algorithms is quite distinct. Appendix A.5 shows some examples of the difference between individual frames, and how that influences the global results.

For some frames, it does not really matter what kind of allocation algorithm is used, as there will be very little difference between the performance of the algorithms. This happens when the motion in the frame can accurately be described using the predicted motion vectors. In such cases, no matter how the complexity is distributed, the optimal allocation is automatically attained. It also means that the most differences will be visible in frames where the predictors need a large amount of refinement, such as when the camera is zooming or objects in the scene are accelerating.

For the remainder of this chapter, we will compare the allocation algorithms in various aspects, until we find the one with the best C-R-D performance. We will look at the global results for each sequence, because they will show the performance of the algorithms averaged over many frames and so are more representative for the average performance difference.

4.2.2 Proportional Allocation versus Inversely Proportional Allocation

First off, we will evaluate whether our initial hypothesis was correct, i.e. whether it is more beneficial to allocate more budget to macroblocks with higher distortion, as opposed to macroblocks with lower distortion.

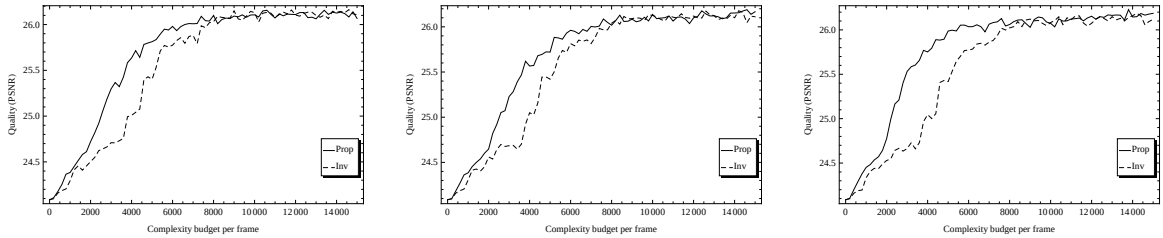
In Figure 4.4, and in detail in Section A.1 of the appendix, we compare the positive and negative allocation variants for all of our distortion metrics. As can be seen, allocation proportional to distortion always yields equal or better quality with respect to inversely proportional allocation.

This means that our original hypothesis was correct: it is better to allocate more to macroblocks with a high distortion, with the goal of decreasing the distortion by doing extra motion estimation and increasing the chances of a good match.

For the remainder of this chapter, we will only consider the proportional allocation strategy.

4.2.3 Direct versus Displaced Distortion

In this section, we evaluate "direct" versus "displaced" distortion. These two algorithms base their allocation on the achieved distortion between the previous source frame and reference frame. The difference is that the direct distortion algorithm uses the distortion between the macroblock in the same location as the current macroblock in the previous frame, while the displaced distortion algorithm uses the distortion in the previous frame of the 16×16 block that is most likely going to be in the position of the current macroblock in this frame.



(a) Direct distortion (b) Displaced distortion (c) Residual

Figure 4.4: Proportional versus inversely proportional allocation for the "Bus" sequence at 500 kbps.

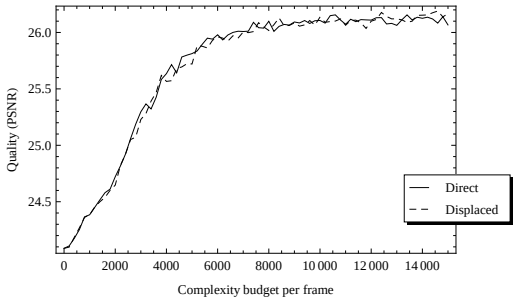


Figure 4.5: Direct versus displaced distortion allocation, for the "Bus" sequence at 500 kbps.

See Section A.2 in the appendix for a full overview of the C-R-D performance of all test sequences. Again, we will show the "Bus" sequence at 500 kbps here as the example (see Figure 4.5).

As can bee seen from the data, there is no appreciable difference in the performance of the algorithms. This is actually unexpected, as it would stand to reason that the displaced distortion would be a better indicator for the current macroblock. There are a couple of reasons this would not be the case:

- The motion vector used to calculate the displacement is not the real motion vector, but an approximation; we use the same motion vector as found in the previous frame. The motion vector can be wrong for the current frame.
- Most motion vectors are not very long; a couple of pixels horizontally and vertically. This means that there is a large overlap between the macroblock's natural position and the displaced block, causing only slightly changed distortion values, which are not large enough to cause a significant difference in allocation.

The second explanation is in our view the most probable reason for this phenomenon.

4.2.4 Direct Distortion Allocation versus Residual Allocation

The residual is the distortion between the macroblock in the previous frame, and the best motion vector candidate in the reference frame before that. It is a measure for

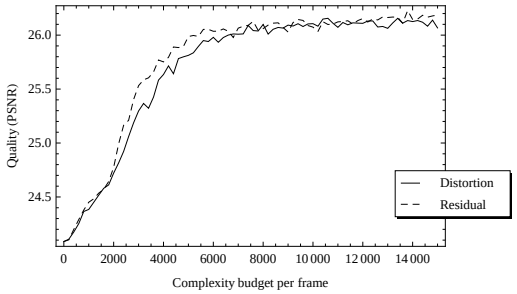


Figure 4.6: Direct distortion versus residual allocation, for the “Bus” sequence at 500 kbps.

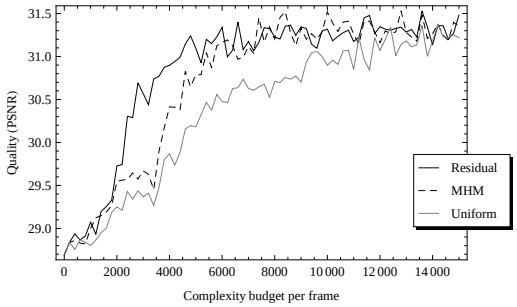


Figure 4.7: Comparison of uniform Allocation, MHM allocation and proportional residual-based allocation, “Bus” sequence at 1500 kbps.

how hard it was to find a good motion estimation match for the macroblock. We have already established that it is better to allocate more complexity to a macroblock if it turns out to be hard to find a match for it (i.e., the size of the residual is large).

Full results are available in Appendix A.3, while Figure 4.6 shows the results for the “Bus” sequence at 500 kbps. The “distortion” allocation shown in this graph is the direct distortion of the previous section. However, since both distortion allocations are practically indistinguishable, we simplify and just call it “distortion”.

Residual-based allocation performs equally and in some cases slightly better to distortion-based allocation. It has the added advantage that the allocation is independent of the quantization factor used by the encoder.

We conclude that the proportional allocation based on the residual metric, is the best of our proposed allocation algorithms. It performs equally well or better than the other proposed algorithms in all cases.

4.2.5 Algorithm Comparison

We now compare the best of our proposed algorithms, Residual-Based Allocation, to Uniform Allocation, and the MHM algorithm introduced in [1]. The full results are available in Appendix A.4. Figure 4.7 shows the most distinctive results for a single video.

We can clearly see that MHM allocation performs equally or better than Uniform Allocation, and residual allocation performs equal to or better than MHM allocation. The

difference is hard to quantify as it highly depends on the video, the bitrate and the complexity bound, but can be as much as up to 1 dB PSNR at 30 dB for the "Bus" sequence encoded at 1500 kbps.

Since we can substitute our residual-based allocation algorithm any place where the MHM algorithm is used, and the C-R-D performance of the encoder would remain the same or improve, we recommend our algorithm in all cases where the other one would be used.

4.3 Conclusions

We have evaluated the budget-based complexity scalability mechanism, and shown that it allows us to bound the complexity of the video encoder to a desired number of clock cycles. Although the scalability mechanism does not have a significant impact on the complexity of the JM encoder, the same mechanism can be applied to commercial H.264 encoders where the impact on complexity consumption will likely be a lot bigger.

Using this scalability mechanism, we have evaluated various complexity budget allocation algorithms for C-R-D performance by encoding a number of videos at a fixed rate and comparing the resulting C-D graphs.

The C-D graphs can be quite noisy. Noise in our measurements is introduced by the dependency of macroblocks on each other, and the independence of successive runs with a different complexity bound. The noise is exacerbated at high quality levels because of the PSNR scale. Still, we can see that quality increases when complexity is increased, up to a certain point.

The quality saturation point comes rather early in the entire scalability range. The sequences we've tested all achieved 95% of their full complexity quality at at most 15% of full complexity. Quality increases quickly with a few steps search in the 16×16 block size, and only marginally after that.

Because the H.264 motion estimation process uses predicted motion vectors, in frames with little or constant motion the predicted motion vector will already be the best motion vector for the macroblock. This means that adding extra complexity will not improve the motion vector, and in these frames the allocation algorithms will perform nearly identically. The difference between algorithms is most prominent in frames where predictors are not as useful, such as zooming and acceleration. This difference does make it harder to see the differences between the algorithms when looking at the global quality of a video.

We have seen that allocation proportional to distortion yields higher quality than allocation inversely proportional to distortion. This confirms our initial hypothesis that high distortion indicates the possibility of improvement when investing extra complexity. With our current allocation method, it does not matter whether the block used for distortion measurement is displaced so it is more accurately measured over the block of pixels that will be in the macroblock's position in the current frame, as the overlap between blocks is very large.

We have seen that it is even better to allocate based on the residual than on the distortion. The residual is a more direct measurement of the effectiveness of motion estimation, and is not influenced by the quantization step of encoding. Because the motion

estimation is the only part we directly control, this allows for a more accurate feedback loop.

Ultimately, our proportional residual-based allocation outperforms both the Uniform Allocation and the MHM Allocation algorithms from literature by up to 1 dB PSNR.

Chapter 5

Conclusion

In this work, we have set the goal to make H.264 encoding complexity scalable. We have done so by introducing a complexity budget, and tying this complexity budget to the number of motion vectors examined during motion estimation. This bounds the number of SAD computations that are done during motion estimation, thereby bounding the complexity consumed by the encoding process of a single frame.

We have shown that the complexity scalability mechanism along with the complexity budget allows us to bound actual complexity, measured in clock cycles, to an arbitrary number within the scalability range of the encoder. Although the scalability afforded by this mechanism does not have a large influence on the total complexity of the JM compiler, the mechanism works and will be more useful in a more optimized H.264 encoder.

We have presented and evaluated three different algorithms intended to allocate the complexity budget of a single frame across its constituent macroblocks, while maximizing the encoder's C-R-D performance. Our algorithms are based on three different measurements of distortion: distortion achieved encoding the current macroblock in the previous frame (direct distortion), distortion achieved encoding the 16×16 block that will be at the position of the current macroblock in the previous frame (displaced distortion), and the best distortion achieved during motion estimation of the macroblock in the previous frame (residual). Of all these algorithms, we've examined both a variant that allocates proportionally to these metrics, and a variant that allocates inversely proportional to them.

We have evaluated these algorithms, and seen that allocating proportionally to distortion yields a better C-R-D performance than allocating inversely proportional.

We have seen that there is no significant difference between the direct distortion allocation and the displaced distortion allocation.

Finally, the residual-based allocation performs equally well or better than the end-to-end distortion-based algorithms in all cases. Additionally, it performs equally well or better than both the Uniform Allocation, where the frame budget is uniformly allocated to all macroblocks, and the MHM allocation introduced in [1].

Although the use of predicted motion vectors in H.264 means the number of frames where the complexity allocation algorithm actually matters is limited, and quality is

quickly saturated with just a small fraction of the possible complexity budget, our proposed Residual-based proportional allocation algorithm should always be used. It's C-R-D performance is equal to or better than all other algorithms in all cases.

5.1 Future Work

We can recommend future work for both our scalability mechanism and the allocation algorithms.

Scalability Mechanism

The mechanism presented in the Section 3.2 is a very simple approach to making motion estimation with mode decisions complexity scalable. Our scalability mechanism is fixed, but it is possible that better results may be achieved by dynamically adapting the scheme to conditions encountered during encoding; for example, allocate more complexity to examine smaller macroblock partitions if it seems likely that the macroblock will eventually be encoded in a small partitioned mode instead of a large one.

As those modifications would necessarily be heuristic, they would need extensive testing. We recommend them as future work.

Some possible avenues of improvement are:

- Do not inspect the modes in the fixed order presented in this work, but vary the order in which the modes are inspected to increase the likelihood of a good match early on in the search process. This is done in [3], where the inspection order of the modes of the current macroblock are based on the modes of the neighboring macroblocks, and in [13] where the order is determined by the R-D cost progression of the major block modes (16×16, 8×8 and 4×4).
- Intra prediction is a different area of mode decisions involving search. This can be made complexity scalable in a similar vein to our inter prediction scalability. Then macroblock budget needs to be allocated between intra prediction and inter prediction, in such a way that the budget is used in the best way.
- Do not spend the budget linearly over all the blocks in a given mode until it is exhausted, but divide it up. Right now, if the budget runs out while estimating some submacroblock that is not the last block of the mode, all remaining blocks will have infinite cost (since no cost calculation at all can be done for them). This will make it very unlikely that the mode will be chosen for encoding. Since this only happens in the last mode that is examined, it remains to be seen if this has a significant impact.
- Right now, the motion estimation budget is spent linearly through all phases of the UMHS algorithm. It may be desirable to skip some phases of the algorithm and go immediately to to refinement when it is determined that the budget is about to run out. As with the previous concern, this will only occur in the last macroblock, but it may be an advantageous strategy when the budget is not spent linearly but allocated across blocks and modes (as in the previous suggestion).

- In our mechanism, we are only treating fullpixel motion search. Subpixel search yields much better results, but when complexity scalable subpixel motion search is introduced, there must be some allocation algorithm for managing the trade-off between budget invested in subpixel refinement, and budget invested in other modes.

Allocation Algorithms

Our allocation algorithm right now uses the gathered metrics directly as macroblock weights in the budget allocation algorithm. This introduces a coupling that is both unnecessary and may hamper quality.

During our research, we have had some interesting but inconclusive results using the distortion metrics not directly as weights, but as a way to rank macroblocks for assignment from a fixed set of weights. Unfortunately, we were unable to pursue this train of research due to lack of time. Our ideas and initial results are included in this thesis as Appendix B in the hopes that they will helpful to future research.

Furthermore, additional work may be done to incorporate this scalability mechanism and frame allocation algorithms into a larger system. The allocation of complexity from GOPs to frames is an open topic, including re-use of unused frame complexity.

Furthermore, our initial goal was to also do work regarding complexity budget allocation between multiple encoders. In the same way as complexity allocation among macroblocks can be tweaked to optimize quality, allocation of a "video encoding" budget to more than one simultaneous encoder can be optimized for global quality.

Appendix A

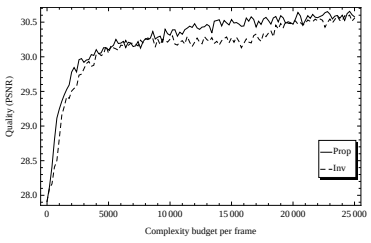
Allocation Algorithm Results

This appendix contains a list of all graphs for all tested video sequences. We have concentrated on high-motion videos. See table A.1 for a list of video sequences tested, their total and average unconstrained complexity consumption, and the unconstrained quality achieved. All video sequences are available on the internet [12].

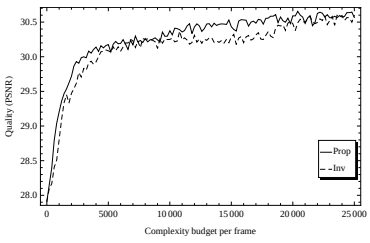
A.1 Proportional versus inversely proportional allocation

In this section, we show the proportional and inversely proportional variants of every distortion-based allocation algorithm for every sequence that we examined.

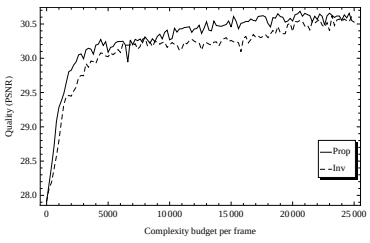
As can be seen, proportional allocation performs equal or better than inversely proportional allocation in all cases. The most difference can be observed in the “Bus” and “City” sequences. “Husky” does not show a lot of difference, and “Stefan” and “Bus” at 6000 kbps are too noisy.



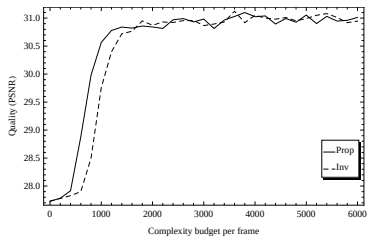
Foreman, Direct Distortion



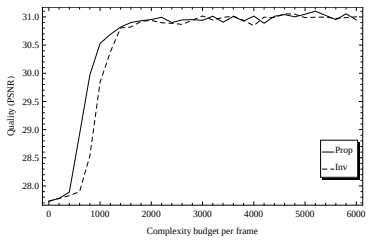
Foreman, Displaced Distortion



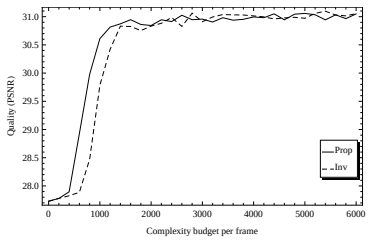
Foreman, Residual



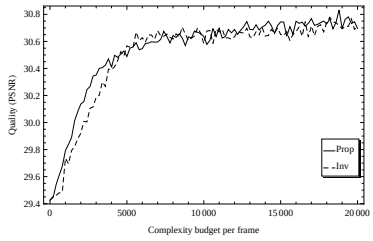
City, Direct Distortion



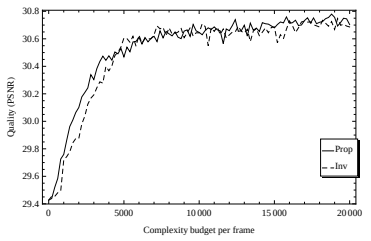
City, Displaced Distortion



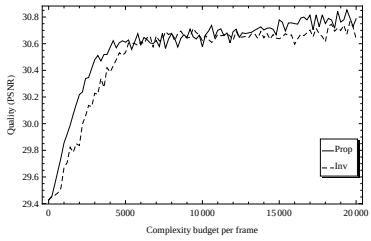
City, Residual



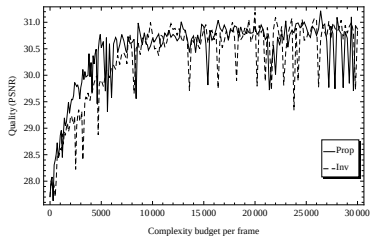
Crew, Direct Distortion



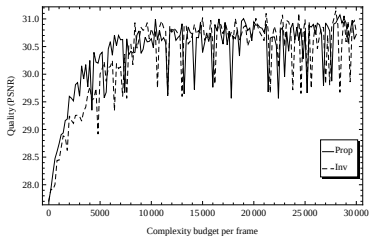
Crew, Displaced Distortion



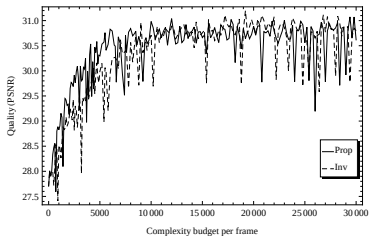
Crew, Residual



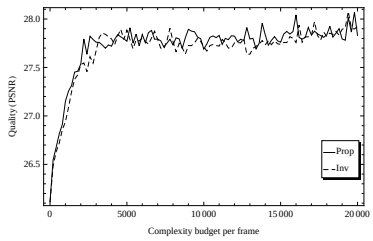
Stefan, Direct Distortion



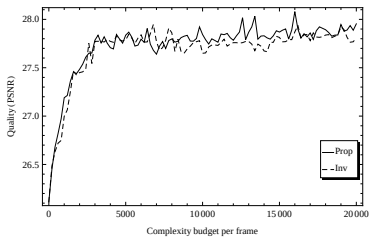
Stefan, Displaced Distortion



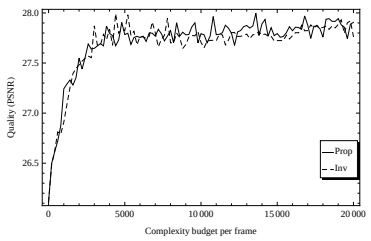
Stefan, Residual



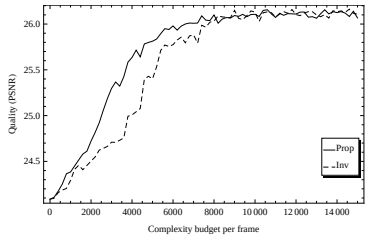
Husky, Direct Distortion



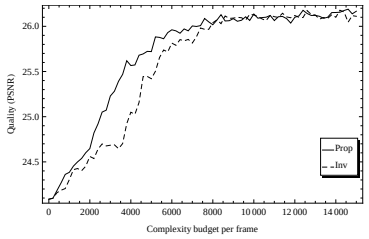
Husky, Displaced Distortion



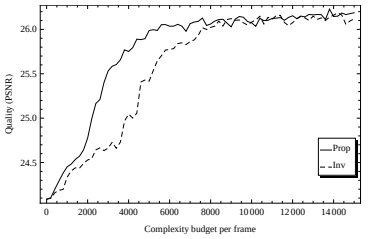
Husky, Residual



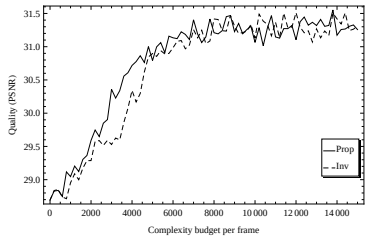
Bus (500 kbps), Direct Distortion



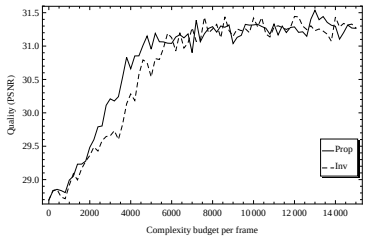
Bus (500 kbps), Displaced Distortion



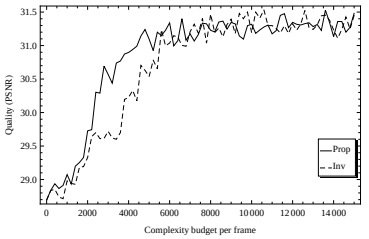
Bus (500 kbps), Residual



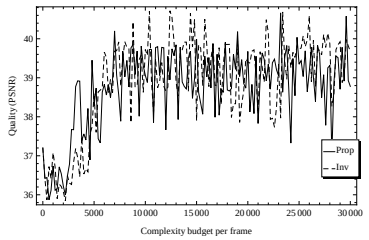
Bus (1500 kbps), Direct Distortion



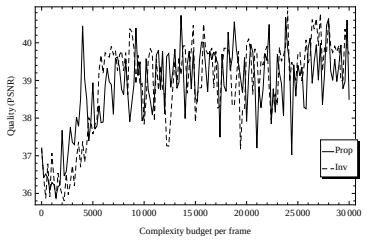
Bus (1500 kbps), Displaced Distortion



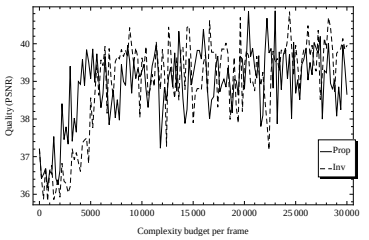
Bus (1500 kbps), Residual



Bus (6000 kbps), Direct Distortion



Bus (6000 kbps), Displaced Distortion



Bus (6000 kbps), Residual

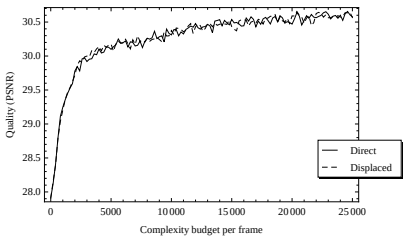
A.2 Direct versus Displaced Distortion

In this section, we show the results of Direct Distortion versus Displaced Distortion. All are using proportional allocation.

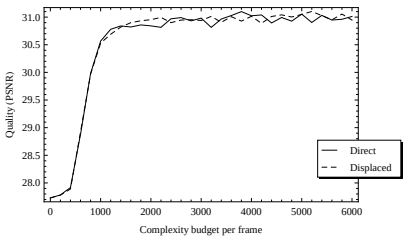
There is virtually no difference between the two allocation algorithms. Because in practice, the displacement is not very large, there is a lot of overlap of the examined areas of the image, and the obtained differences in distortion are not significant enough to cause large changes in allocation.

Sequence	Frames	Bitrate	Classification	Avg complex.	Quality
Foreman	300	250	Medium motion	$1.9 \cdot 10^5$	30.84 dB
City	600	500	Medium motion	$1.7 \cdot 10^5$	31.20 dB
Crew	600	250	Medium motion	$2.8 \cdot 10^5$	31.00 dB
Stefan	300	1500	High motion	$2.1 \cdot 10^5$	31.06 dB
Husky	250	4000	High motion, fine detail	$3.6 \cdot 10^5$	28.20 dB
Bus	150	500	Very high motion	$3.5 \cdot 10^5$	26.57 dB
Bus	150	1500	Very high motion	$2.5 \cdot 10^5$	31.64 dB
Bus	150	6000	Very high motion	$1.9 \cdot 10^5$	38.46 dB

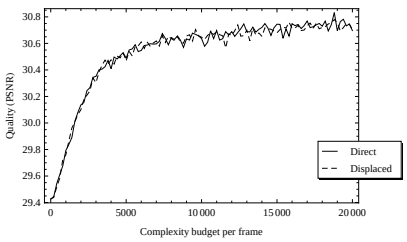
Table A.1: The list of testing sequences used and the rates tested. Each video is encoded at 30 frames per second. All videos are CIF resolution (352×288) with the exception of "Stefan", which is SIF resolution (352×240). Bitrate is in kbps, Quality is in dB PSNR. Average complexity is the average unbounded complexity consumption of each frame in the sequence, while Quality is the quality achieved at the given rate without bounding complexity.



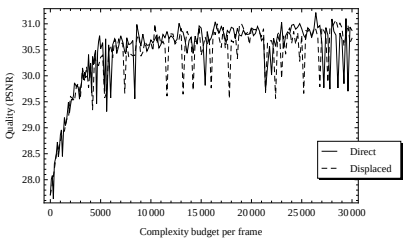
Foreman



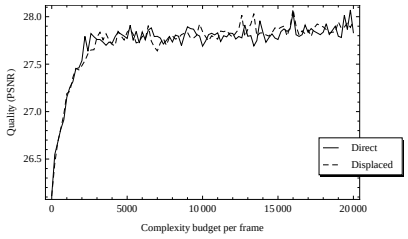
City



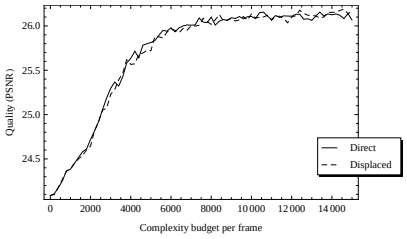
Crew



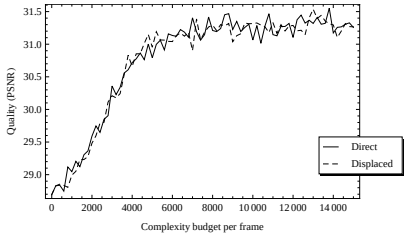
Stefan



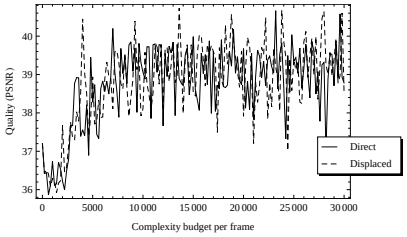
Husky



Bus (500 kbps)



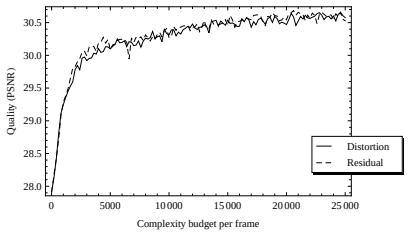
Bus (1500 kbps)



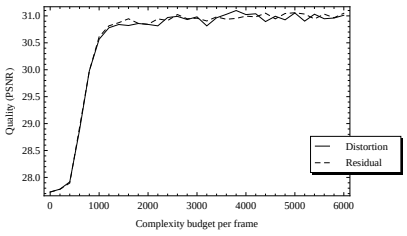
Bus (6000 kbps)

A.3 Direct Distortion versus Residual

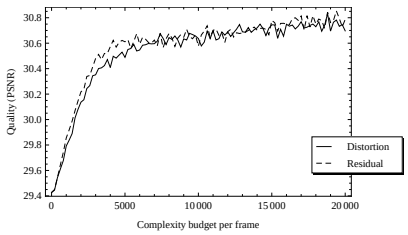
This section shows the result of the comparison between direct distortion-based allocation and residual-based allocation. Both are using proportional allocation. Residual-based allocation is equal in all cases, superior in the “Bus” sequence and slightly superior in the “Crew” sequence. Again, “Bus” at 6000 kbps and “Stefan” are too noisy to base conclusions on. However, we can see that residual-based allocation is always equal or superior to distortion-based allocation, so should be used in all cases.



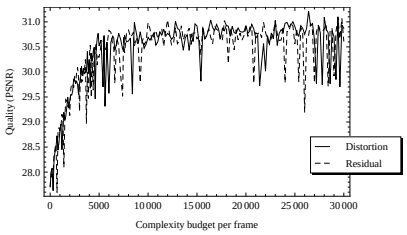
Foreman



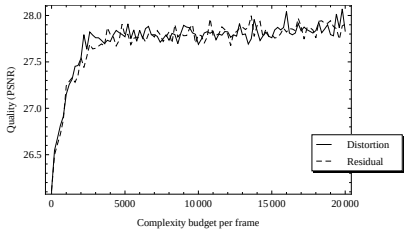
City



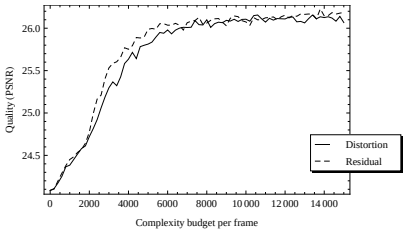
Crew



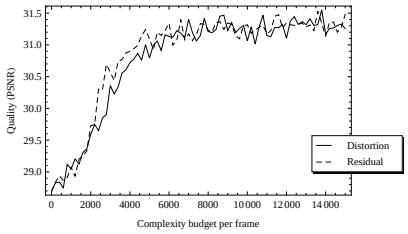
Stefan



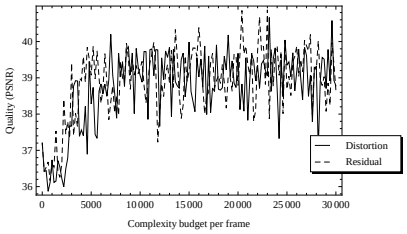
Husky



Bus (500 kbps)



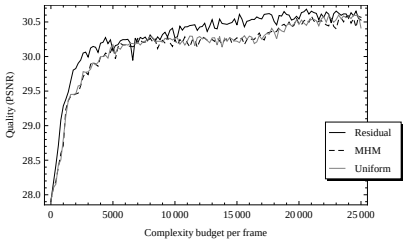
Bus (1500 kbps)



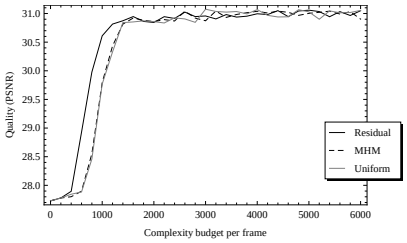
Bus (6000 kbps)

A.4 Residual, MHM and Uniform

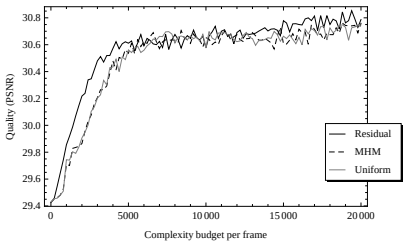
This section shows the comparison between Uniform Allocation, MHM allocation and the best of our proposed algorithms, residual-based proportional allocation. Clearly, the Residual-based allocation is always equal or superior to the other allocation methods. Even in the noisy data of “Stefan” and “bus” at 6000 kbps this can be discerned.



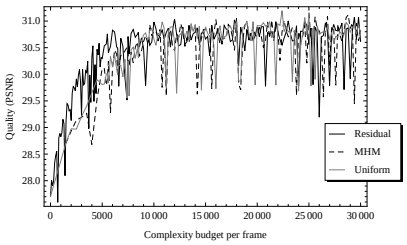
Foreman



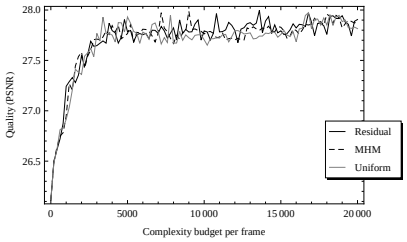
City



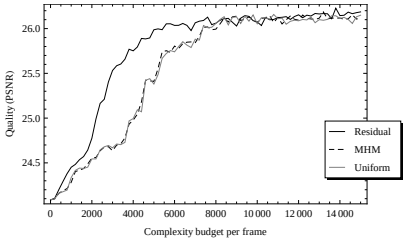
Crew



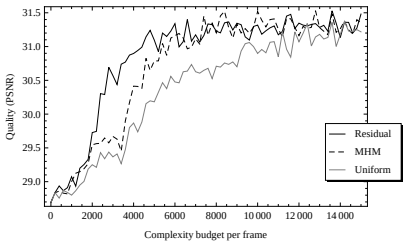
Stefan



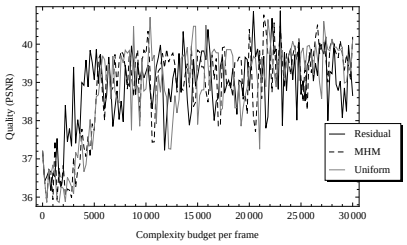
Husky



Bus (500 kbps)



Bus (1500 kbps)

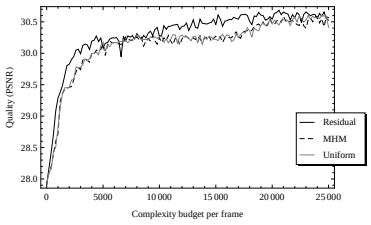


Bus (6000 kbps)

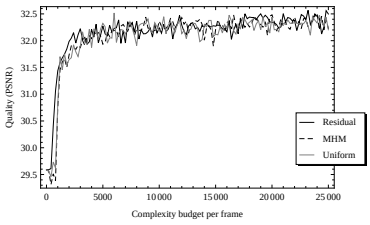
A.5 Individual Frames versus Global Results

This section shows the distinction between the global results for each video, versus the results for individual frames. For each sequence, we show the global results, the results for a frame with very little difference between the algorithms and a frame with a lot of difference between the algorithms.

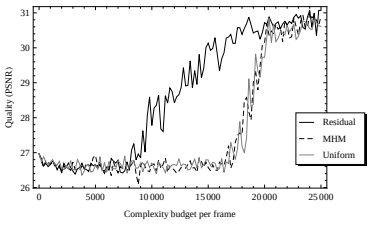
We can see that there is a lot of difference among the frames of one sequence, depending on how easy it is to motion-estimate the frame, and how well the predicted motion vectors describe the actual motion in the scene. The partial ordering of the algorithms is consistent however: every frame shows that one algorithm consequently matches or outperforms another, independent of the accuracy of the results.



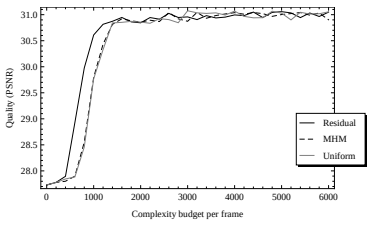
Foreman global



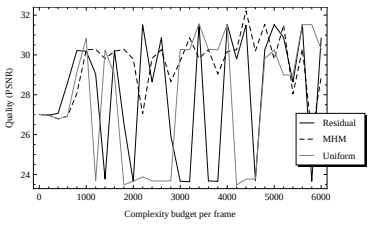
Foreman frame 72



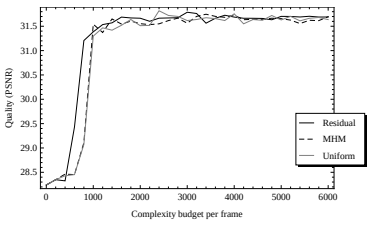
Foreman frame 217



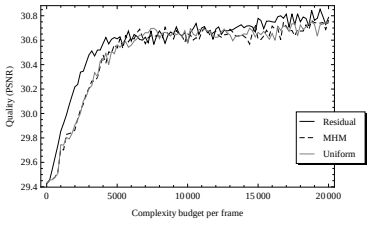
City global



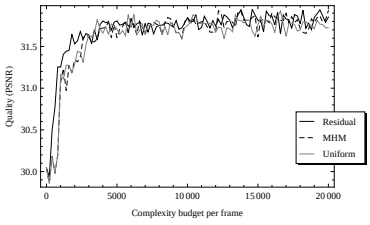
City frame 61



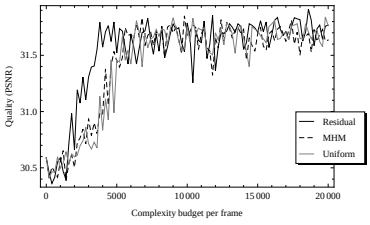
City frame 192



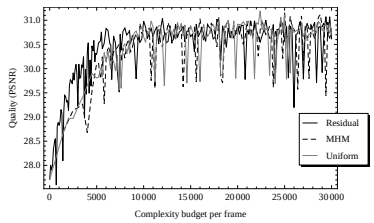
Crew global



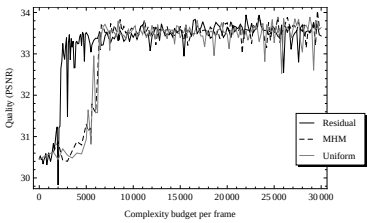
Crew frame 48



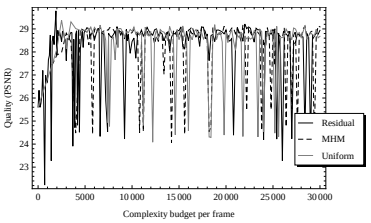
Crew frame 229



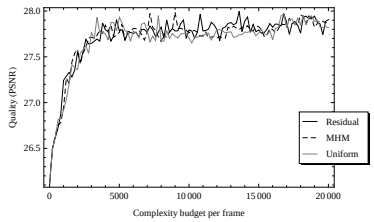
Stefan global



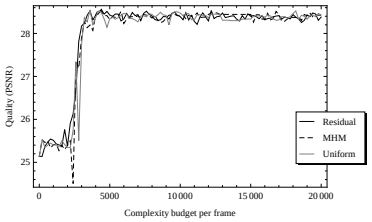
Stefan frame 221



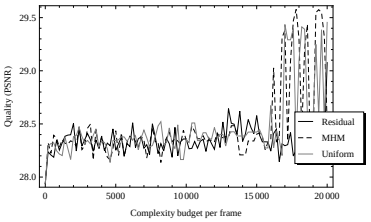
Stefan frame 286



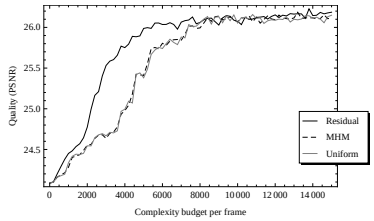
Husky global



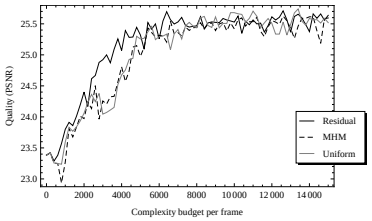
Husky frame 115



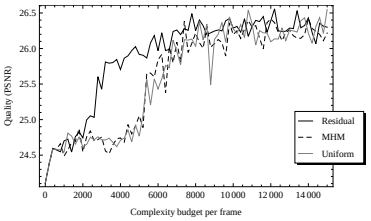
Husky frame 167



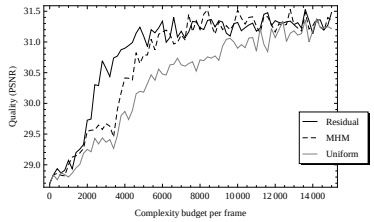
Bus (500 kbps) global



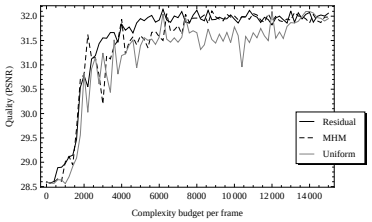
Bus (500 kbps) frame 52



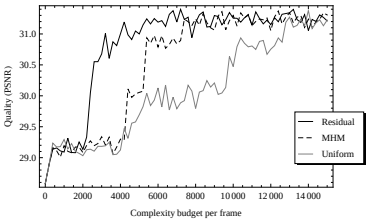
Bus (500 kbps) frame 100



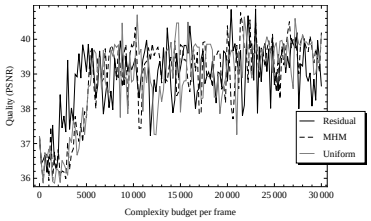
Bus (1500 kbps) global



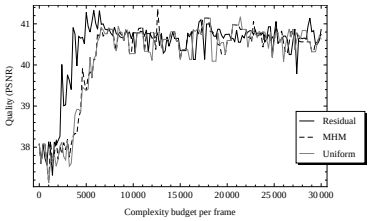
Bus (1500 kbps) frame 76



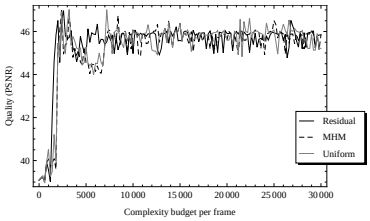
Bus (1500 kbps) frame 94



Bus (6000 kbps) global



Bus (6000 kbps) frame 17



Bus (6000 kbps) frame 75

Appendix B

Fixed Weight Allocation Strategy

Our distortion-based allocation algorithms proposed in Section 3.3 have one thing in common: the value measured for distortion is directly used as a weight in the proportional allocation strategy. This introduces a coupling that is unnecessary and may be harmful to efficiency. We have done some work investigating an allocation algorithm where the measured distortion and assigned weights are decoupled, which is presented here with the hope that it will be useful to future researchers.

B.1 Algorithm

In this proposed algorithm, we continue to estimate the complexity needs of a macroblock based on a distortion metric from the previous frame. However, rather than allocation complexity proportional to the measured distortion, we use the distortion to sort the macroblocks by order of complexity need. We then assign each macroblock a weight, taken in order from a fixed set of weights, such that a macroblock with a higher distortion will have a higher weight, but not necessarily directly related to the distortion itself.

B.1.1 Motivation

From our research in this thesis, it is clear that a higher distortion indicates a larger need for complexity. However, it is not clear that the relative distortion among macroblocks has a direct bearing on the relative complexity needs. A way to decouple the complexity need identification and the budget assignment is desired.

By sorting by distortion, and assigning weights from a non-increasing distribution, we achieve the goal that blocks with higher distortion receive more complexity, while maintaining the freedom to modify the distribution as we see fit, to better adapt to the video being encoded.

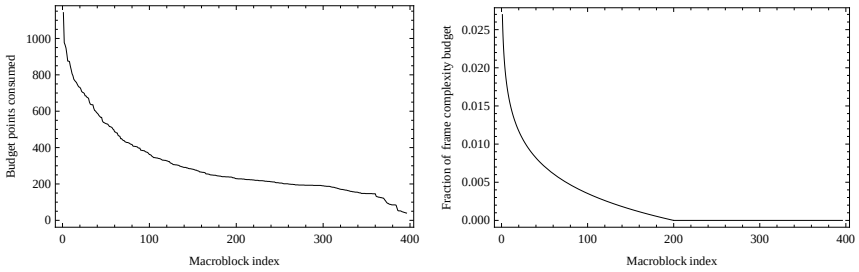


Figure B.1: (a) Sample macroblock complexity consumption distribution, taken from frame 115 of the "Foreman" sequence, in complexity budget points. The macroblocks have been ordered from most to least complexity consumption. (b) Restricted logarithmic distribution used in experiments, in fraction of the frame budget. There are 396 macroblocks in the CIF format.

B.1.2 Weight Distribution

The weight distribution was originally derived from the natural distribution of complexity budget consumption of an unbounded run of a video. Figure B.1(a) shows the unconstrained macroblock budget consumption for a frame of the "Foreman" sequence.

We tried matching this distribution using a logarithmic function, but this yielded too many macroblocks with a similar budget allocation, leading to very little difference in allocation and quality very similar to the Uniform Distribution. That is why we removed the tail of the distribution, allocating complexity only to the upper half of the sorted list of macroblocks. The distribution that we used for the experiment is shown in Figure B.1(b). We have not done research into the parameters of this distribution, but for videos of CIF resolution the shape of the distribution is produced using the following formula:

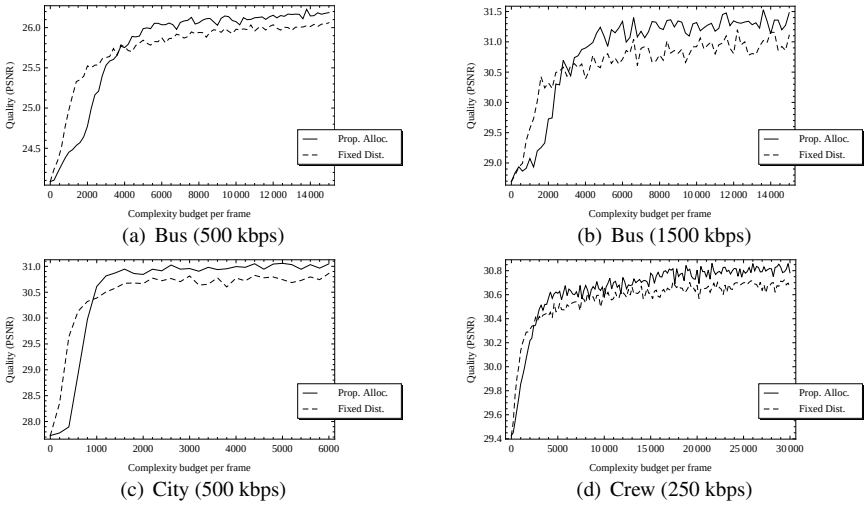
$$w(i) = 0.3 - \frac{log_{10}(5i)}{10} \uparrow 0 \tag{B.1}$$

We then normalize the values of the distribution such that they sum to 1 over the entire number of macroblocks M, and multiply with the frame budget to obtain the macroblock budgets. Let $ord(m,n)$ return the index of macroblock m in frame n if the macroblocks are sorted descending by metric value $x_{m,n}$, $1 \leq ord(m,n) \leq M$. the macroblock budget is calculated as follows:

$$B_{m,n} = \frac{w(ord(m,n-1))}{\sum_{1 \leq i \leq M} w(i)} \cdot B_{frame}(n,m) \tag{B.2}$$

B.2 Preliminary Results

We use the same metric for the fixed weight distribution allocation as the best algorithm presented in the main part of this thesis, which is the Residual, and encoded the same



(a) Bus (500 kbps) (b) Bus (1500 kbps)

(c) City (500 kbps) (d) Crew (250 kbps)

Figure B.2: Proportional allocation versus allocation based on the fixed distribution for a number of sequences. Both allocations are based on the "Residual" metric.

sequences at the same bitrates. Figure B.2 shows the two allocation strategies side-by-side.

An interesting result can be observed: when the complexity budget is highly constrained, the new fixed distribution-based allocation yields higher quality. However, after a certain tipping point, which varies per video, the proportional allocation yields higher quality.

It seems that when the budget is restricted, it is better to focus most of this budget into a few macroblocks that indicate the highest complexity need, in an attempt to get the most improvement out of these macroblocks and thereby improving the overall quality of the video.

On the other hand, our distribution does not compensate for the fact that when the budget grows, it is grossly overallocating complexity to those few macroblocks while still starving the other half of the macroblocks.

Nevertheless, the allocation strategy seems promising. The low-complexity results are distinctly better and the distribution is highly modifiable. Future research is recommended to address the complexity overallocation at higher budgets, which may lead to similar or even higher quality than the proportional allocation. One possible approach could be stretching the shape of the distribution based on the complexity budget, so the tail becomes longer at higher budgets and the blocks at the far end are no longer starved.

Another thing that should be considered is that during our experiments we only measure objective quality. It could be that at low complexity bounds, even though the objective quality is higher, the subjective quality could be lower since most quality improvement is focused into a select number of macroblocks.

Appendix C

Glossary

This appendix contains an overview of the terms, abbreviations and symbols used in this thesis.

C.1 List of Terms

Term	Clarification
Complexity	The time needed for a computer to execute a program. Can be measured in instructions, clock cycles or wall-clock time.
Complexity Budget	The maximum amount of time that a piece of software is allowed to take. *(In this work)* The number of vectors that are allowed to be examined during Motion Estimation.
Clock Cycle	Discrete unit of time in which a CPU executes instructions.
Decoder	A piece of software that can decode an encoded bitstream according to a specific video format, and reconstruct the original uncompressed video frames.
Distortion	The difference in pixel values between a source video frame (or block of pixels) and the encoded and decoded version of the same frame (or block of pixels). Ideally the distortion should be low.
Encoder	A piece of software that compresses a sequence of uncompressed video frames into an output bitstream.
Frame	A single picture in a video sequence.
Frame Rate	The temporal resolution of the video in Hertz.
Hadamard Transform	A transform similar to the Discrete Fourier Transform (DFT) or Discrete Cosine Transform (DCT), but using Walsh functions instead of sines.

Term	Clarification
Macroblock	A 16×16 block of pixels. Fundamental unit of video encoding. Every frame is partitioned into macroblocks, each of which is encoded separately.
Motion Estimation	The process of finding the displacement of a macroblock with respect to the previous frame.
Motion Vector	Vector that indicates the displacement of a macroblock with respect to the previous frame. It is found using Motion Estimation.
Output Bitstream	The encoded representation of a video.
Rate	The number of bits needed to encode the video or a piece of the video. Ideally the rate should be low.
Rate-Distortion Performance	An evaluation of the combined rate and distortion achieved by the encoder for a certain video sequence or a piece of the video sequence. Ideally the encoder should achieve both a low rate and a low distortion.
Reference Frame	The encoded and decoded version of a frame (so it is the same as the frame received and decoded by the decoder), which is used for motion estimation by the encoder.
Quantization	The process of reducing the absolute values in the coefficient matrix of a macroblock, by dividing them with a constant Q, the quantization parameter.
Variable Length Coding	A lossless compression technique where common sequences of bits in the input stream are represented with short bitsequences in the output stream so they can be represented efficiently.

C.2 List of Abbreviations

CABAC Context-Adaptive Binary Arithmetic Coding

CAVLC Context-Adaptive Variable Length Coding

CE Consumer Electronics

CPU Central Processing Unit

C-R-D Complexity-Rate-Distortion

DCT Discrete Cosine Transform

GOP Group of Pictures

ISO International Standards Organization

ITU-T Telecommunication Standardization Sector

ME	Motion Estimation
MHM	Motion History Matrix
MPEG	Moving Pictures Expert Group
MV	Motion Vector
PMV	Predicted Motion Vector
PSNR	Peak Signal-to-Noise Ratio
R-D	Rate-Distortion
SAD	Sum of Absolute Differences
SSD	Sum of Squared Differences
TSC	Timestamp Counter
UMHS	Uneven Multi-Hexagon Search
VLC	Variable Length Coding

C.3 List of Symbols

Symbol	Meaning	Page
α	Conversion factor from complexity measured in budget points to complexity measured in clock cycles.	24
β	Required complexity associated with non-scalable processing operations of a macroblock, in clock cycles.	24
γ	Complexity overhead for processing a frame, in clock cycles.	24
B_{frame}	Complexity budget of a frame, in 16×16 SAD computations.	24
$B_{frame}(n)$	Complexity budget of frame n.	28
$B_{frame}(n,m)$	Remaining complexity budget of frame n before processing macroblock m.	28
$B_{m,n}$	Complexity budget of macroblock m in frame n	27
$B^{\rho}_{m,n}$	Complexity budget of macroblock m in frame n determined using proportional allocation.	27
$B^{\iota}_{m,n}$	Complexity budget of macroblock m in frame n determined using inversely proportional allocation.	28
$\overline{C}_{m,n}$	Actual consumed complexity of macroblock m in frame n, in budget points.	28
C_{frame}	Complexity of encoding a frame, in clock cycles.	24

Symbol	Meaning	Page
$cost(\mu)$	R-D cost of a macroblock if it is encoded in mode μ.	13
$D(\mu)$	Distortion obtained encoding the macroblock in mode μ.	13
F_μ	Ratio of the area of a submacroblock in mode μ to the area of a 16×16 block.	24
λ	Weighing factor of rate versus distortion.	13
M	Number of macroblocks in the frame.	24
N	Number of frames in the sequence.	27
μ	Symbol that represents an encoding mode.	13
Q	Quantization factor of a macroblock.	11
$R(\mu)$	Rate requirements for encoding the macroblock in mode μ.	13
S	Maximum value of a sample. Either 255 for 8-bits color information or 65535 for 16-bits color information.	7
$x_{m,n}$	Metric value used in complexity allocation for macroblock m in frame n.	27

Bibliography

[1] Z. He, Y. Liang, L. Chen, I. Ahmad, and D. Wu, "Power-rate-distortion analysis for wireless video communication under energy constraints," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 15, no. 5, pp. 645–658, 2005.

[2] T. Wiegand, G. Sullivan, G. Bjontegaard, and A. Luthra, "Overview of the H. 264/AVC video coding standard," *IEEE Transactions on circuits and systems for video technology*, vol. 13, no. 7, pp. 560–576, 2003.

[3] L. Su, Y. Lu, F. Wu, S. Li, and W. Gao, "Real-time video coding under power constraint based on H.264 codec," *Visual Communications and Image Processing 2007. Edited by Chen, Chang Wen; Schonfeld, Dan; Luo, Jiebo. Proceedings of the SPIE*, vol. 6508, p. 650802, 2007.

[4] S. Mietens, P. H. N. de With, and C. Hentschel, "New complexity scalable MPEG encoding techniques for mobile applications," *EURASIP J. Appl. Signal Process.*, vol. 2004, pp. 236–252, 2004.

[5] J. Ostermann, J. Bormans, P. List, D. Marpe, M. Narroschke, F. Pereira, T. Stockhammer, and T. Wedi, "Video coding with H. 264/AVC: tools, performance, and complexity," *IEEE Circuits and Systems Magazine*, vol. 4, no. 1, pp. 7–28, 2004.

[6] A. Tourapis, O. Au, M. Liou, *et al.*, "Predictive motion vector field adaptive search technique (PMVFAST)-enhancing block based motion estimation," in *proceedings of Visual Communications and Image processing*, vol. 2001, San Jose, CA: IEEE Proceedings of SPIE, 2001.

[7] Z. Chen, P. Zhou, and Y. He, "Fast integer pel and fractional pel motion estimation for JVT," *JVT-F017, December*, 2002.

[8] Z. Chen, P. Zhou, and Y. He, "Fast motion estimation for JVT," *JVT-G016. doc, Joint Video Team (JVT) of ISO/IEC MPEG & ITU-T VCEG*, 2003.

[9] M. Zhou and T. Incorporated, "Scalable Intra Prediction," in *document JVT-C033 doc, Joint Vide Team (JVT) of ISO/IEC MPEG & ITU-T VCEG Meeting Fairfax, Virginia, USA*, pp. 6–10.

[10] S. Saponara, C. Blanch, K. Denolf, and J. Bormans, "The JVT advanced video coding standard: complexity and performance analysis on a tool-by-tool basis," in *Proc. 13th Int. Packetvideo Workshop*, pp. 98–109, 2003.

[11] M. Pettersson, "Linux Performance Counters Driver." `http://user.it.uu.se/~mikpe/linux/perfctr/`.

[12] http://media.xiph.org/video/derf/. Xiph.org Test Media.

[13] P. Yin, H.-Y. Tourapis, A. Tourapis, and J. Boyce, "Fast mode decision and motion estimation for JVT/h.264," in *Image Processing, 2003. ICIP 2003. Proceedings. 2003 International Conference on*, vol. 3, pp. III–853–6 vol.2, Sept. 2003.