Improving multipoint video conferencing using scalable video coding

Dekkers, M.

Award date:
2006

TECHNISCHE UNIVERSITEIT EINDHOVEN
Department of Mathematics and Computer Science

Improving Multipoint Video Conferencing using Scalable Video Coding

By
M. Dekkers

Supervisors:

Arjen Klomp (LogicaCMG)
Reinder Bril (TU/e)

Eindhoven, June 2006

Improving Multipoint Video Conferencing using Scalable Video Coding

LogicaCMG
Kennedyplein 248
5611 ZT Eindhoven

Technische Universiteit Eindhoven (TU/e)
Den Dolech 2
5612 AZ Eindhoven

Contractor / supervisor LogicaCMG:
ir. A.L. Klomp
Industry, Distribution and Transport
BU Technical Software Engineering
arjen.klomp@logicacmg.com

Supervisor TU/e:
dr.ir. R.J. Bril
Department of Mathematics and Computer Science
System Architecture and Networking (SAN)
r.j.bril@tue.nl

Author:
ing. M. Dekkers
Department of Mathematics and Computer Science
Student nr: 462677
Graduation: June 2006
mdekkers@ramenta.nl

logicaCMG

TU/e technische universiteit eindhoven

Preface

"Luctor et Emergo"

Here it is, my master's thesis, the result of a nine month period that concludes my study Computer Science, with specialization Embedded Systems, at the Technische Universiteit Eindhoven (TU/e).

The work for this project has been done at LogicaCMG Eindhoven, an international operating consultancy company with a mission to help leading organizations worldwide achieve their business objectives through the innovative delivery of information technology and business process solutions. I worked at the business unit 'Technical Software Engineering', which is part of the division 'Industry, Distribution and Transport'.

My project aimed at improving a multipoint video conferencing on a resource constrained platform. Two keywords of this project are *scalability* and *adaptation*, referring to scalable video coding and choosing a GOP size adaptively. These words can also be applied personally. Within the context of my research I *scaled* my scope of view from the vision of a 'fully connected world' to a small detail within a video encoder. Secondly, during my project I constantly had to *adapt* in order to cope with the challenges and uncertainties part of a graduation.

I would like to thank Reinder Bril and Arjen Klomp for their technical supervision and guidance during my graduation. For all the non–technical support I needed, the love of my girlfriend Marlous was invaluable. I'd also like to thank the packaging team of LogicaCMG for their exclusive lunches. And finally Kai, who's always purring when I come home, putting a smile on my face regardless what day I had.

Maarten Dekkers
Eindhoven, 24th April, 2006.

(The dictionary–like quotes used in this thesis are the result of a keyword search through google.com, dictionary.com, wikipedia.org and wiktionary.org)

Summary

In this thesis, options for improving the quality of a multipoint video conference are studied. A multipoint video conference is a discussion between three or more (groups of) people who are in different places, but can see and hear each other using electronic communications. The context of this study is a fully connected world where different kinds of terminals, such as fixed/mobile phones, set–top boxes and personal computers are all interconnected. These terminals have constrained network and processing resources, but due to the heterogeneity of the terminals the available resources vary.

During a multipoint video conference several video streams are received by a participant, but a terminal can only receive, decode and combine a limited amount of streams in real–time. The question is: How to utilize the available resources as efficient as possible and improve the quality?

A solution is presented that uses scalable video coding to adapt the quality to the available resources. A scalable video encoder creates a bit–stream that allows the extraction of a lower quality layer, requiring less resources to decode or send. This can be a lower framerate, picture size of picture quality. Mapping these layers to several multicast channels allows a terminal to 'tune in' to a quality level that matches its available resources as closely as possible. This optimizes available bandwidth, while serving heterogenous terminals.

The notion that in a multipoint conference the active speaker should be best visible provides a second option to trade video quality for resources. This is utilized by degrading the quality of the non–speakers and freeing terminal resources that can be used to highlight the active speaker.

In an architectural investigation, technologies behind video conferencing are studied. The connection between participants can be centralized or decentralized, of which the decentralized method is preferable. For transmitting audio and video, unicasting, broadcasting or multicasting is available. Multicasting is preferable, but is currently not widely available. To extract a lower quality level as early as possible, the payload identification feature of the Real-time Transport Protocol (RTP) can be used. The preferred video coding standard for a next–generation video conferencing application is the Advanced Video Codec (AVC), as it is opted to be the next defacto standard.

The Joint Scalable Video Model (JSVM) is an extension to AVC, providing scalable video coding. Only a slow reference implementation of JSVM is currently available that cannot be used in an application with real–time requirements. A demonstration program has been created that shows the feasibility of using scalable video to optimize the use of available resources and highlighting the active speaker. The program uses a created model of the available processing resources and the resources needed to decode a specific quality level.

The demonstration program shows that the visible effect of using spatial scalability for highlighting the active speaker is limited. As it also has a large impact on required resources, this makes it less suitable for optimizing processing resources locally. It can be used to serve heterogenous terminals. For providing local optimization, temporal and quality scalability can be used to highlight the active speaker by degrading the quality of the non–speakers.

To create a video conferencing application an optimized version of JSVM is needed. To utilize the provided scalability, the resources required for decoding different quality layers should be determined. This highly depends on the target platform.

A separate chapter in this thesis is a proposal for improving a specific algorithm of the JSVM codec. This algorithm allows the adaptive creation of a bit–stream with variable Group of Pictures (GOP) sizes. The proposed algorithm results in an improved picture quality/coding efficiency by creating an optimal division in the possible sub–GOPs. Determination of the impact of this improvement is still future work.

Contents

List of figures

Chapter 1

Introduction

in·tro·duce (vb): "To provide (someone) with a beginning knowledge or first experience of something."

1.1 Context

The world of communication is undergoing a constant change. At the moment, most telephone calls are transferred through an analogue line and also the television signal is received through an analogue connection. Both of these signals are significantly different and are not inter-operable. A telephone cannot show a television signal or vice versa. The rise of fast internet connections allows these analogue signals to be sent digitally to the users. The telecommunication market is already gradually changing to transporting phone calls digitally, known as Voice over IP (VoIP). Also the transition from analogue to digital television is expected to get an enormous boost with this years FIFA world championship. The advantage of these digital signals is that they can all be transported using the same broadband internet connection. Because the same network is used to send these signals, different kinds of terminals, such as computers, fixed/mobile phones and televisions, are connected to each other. Connecting these heterogenous terminals using one network forms an idealized vision of a *fully connected world*, as shown in figure 1.1.

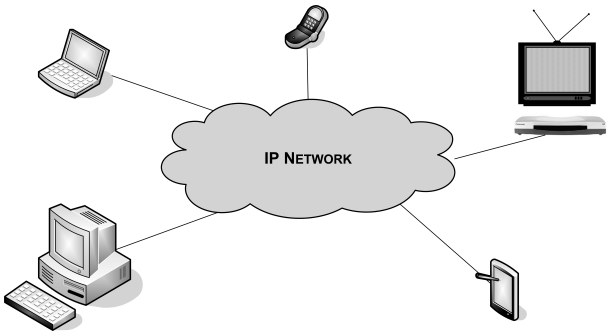


Figure 1.1: The fully connected world

The *fully connected world* opens a wide range of new possibilities. For example, it becomes possible to have a (video)phone call or send text/multimedia messages between a mobile phone and television. Another interesting option, from a user perspective, is the ability to store personal data on a central server in the network. This data, such as an address book, can be accessed from every different terminal. At the moment LogicaCMG is devel-

oping a framework for providing services in this *fully connected world* called uOne™ [16]. uOne™ allows the central storage of an address book and the starting of a service, such as messaging, from this address book from different terminals.

In order to connect a television (that only accepts analogue signals) to a digital network, a set–top box is needed. Such a set–top box translates the received digital signals to video signals that can be shown on the television. Most set–top boxes are hardware platforms with limited memory/processing power and with specialized hardware for signal processing. The current set–top boxes are mostly used to receive and translate digital television signals, a one way connection from provider to consumer.[1] The only two way interaction is used for 'video-on-demand', where a movie can be rented using the digital connection. The next step is to use the television as a two way communication device, for instance in a (video)phone call. For such an application, the set–top box has to do more than translate the incoming digital signals. A camera and microphone have to be connected and the audio and video from these devices must be digitized. Next to this capturing, more elaborate network options are needed to be able to send the audio and video. Such an application is within the interest of the competence center Technical Software Engineering (TSE) from LogicaCMG as TSE focusses on embedded software engineering. A generic personal computer can be used instead of a set–top box, but this is not recommended, as such a computer is much too powerful, noisy and too large for placing it near a television in the living room.

Next to the previous sketched *fully connected world*, a new technology is rising in the market: the digital video compression standard AVC, also known as H.264. This standard provides high video quality at bitrates substantially lower than previous standards such as MPEG-2. AVC can be used in a broad spectrum of applications, from high quality video broadcasting to low quality mobile video conferencing and is opted to be the next generation video compression standard, replacing MPEG-2. AVC has been adopted for the (mobile) transport of video by major industries, such as those participating in the Digital Video Broadcast (DVB) project and the 3rd Generation Partnership Project (3GPP) [31, 26]. LogicaCMG is interested in the use of AVC on a set–top box.

The combination of the *fully connected world*, providing a service in this world using a set–top box and the use of AVC define the background of this master project.

1.2 Problem definition

To narrow the scope of research, a specific service was chosen within the previous mentioned context. This service is *decentralized multipoint video conferencing* using a television and a set–top box. Video conferencing is an application that poses strict requirements on a target platform with respect to the processing power and memory requirements. To stress things further, *multipoint* video conferencing connects three or more different participants and *decentralized* refers to the combining of the different audio/video streams at the receiver. A set–top box, a resource constrained platform, can only decode and combine a limited amount of different video streams. In such an application the perceived quality becomes an issue. This yields the following research question:

How to improve the quality of a decentralized multipoint video conference on a resource constrained platform, within the context of a 'fully connected world'?

[1] More complex set–top boxes are arriving that provide more functionality combining a broadband internet connection, digital telephone and television.

The focus, when answering this question, is mainly on the processing on a set–top box. It is assumed that the network is not a bottleneck, but network requirements are kept in mind. As stated before, for compressing video the AVC standard is preferable, as it is opted to become the new defacto video compression standard. A third decision was to focus primarily on video only and keep audio out of the scope.

1.3 Current state of video conferencing

For having a video conference in the *fully connected world*, the provided service must be able to operate beyond telecom/cable operator boundaries. This requires that the used technologies should be open or standardized, so that each party can use them. It is not preferable that an operator uses an own (proprietary) technique. Another requirement is that the video conferencing application should be as 'lightweight' as possible, as a set–top box has to provide different services and its available resources are constrained.

Several solutions for multipoint video conferencing currently exist on the market. These can be divided in two types. The first is dedicated video conferencing hardware, the other is video conferencing software used on a personal computer. The drawback of (technologies used in) dedicated hardware is that they mostly use a central server for combining several streams. For the context in mind, a central server is not useful, because such a server is a single point of failure and it is not able to scale to large conferences. This limitation will be explained more elaborately in chapter 2. Secondly the terminals used are, more or less the same with respect to processing/network resources.

For a personal computer (PC), a wide range of video conferencing programs can be found. To serve different users, most of them provide as much functionality as possible by supporting several kinds of video conferencing standards and audio/video compression standards. Generally, the software meant for a PC is not lightweight enough for use on a set–top box.

To optimize services between different users Quality of Service (QoS) has been introduced. QoS trades network and terminal resources to achieve an optimal quality. In [12] scalable video decoding algorithms are used to provide QoS. Another application that uses QoS is GCSVA, a software based, research application that uses frame dropping filters for scaling the video for adapting to different decoding/network capabilities and showing the active speaker at a higher quality [4]. From this solution the notion came that in a multipoint video conference, the active speaker should be best visible. In [4] it is also stated that the used filters can overload less powerful hosts, thus not suitable for a set–top box. The drawback of both mentioned solutions is that they both use an adapted video codec that cannot be used as a generic solution.

In general, the found solutions do not optimally fit in the predefined context. The solution, as will be presented, provides a more flexible approach that efficiently uses the available resources and where heterogenous terminals can achieve as much quality as possible.

1.4 Method of investigation

For this master project, first an assignment was created together with the supervisors. This assignment contained the previously mentioned context and interest in AVC. After creating the assignment, a technology study was performed. This study was needed to build up knowledge on current video conferencing solutions and the different technolo-

gies used. With this knowledge a next–generation video conferencing architecture was proposed. This architecture can be used in the fully connected world with heterogenous terminals. It uses scalable video coding and multicasting to provide QoS adjustment and improve the perceived quality.

The used scalable video encoder is JSVM, a reference implementation of a scalable extension to AVC [21, 22]. During the development of the application, it was found that a program, using a reference implementation, could not be created. The reference codec is far too slow to en- or decode video in real–time. As was discovered, a reference codec is merely used to show an example of how the specified bit–stream can be created and decoded. It can be used to look at the features of a standard and to test added improvements. Due to this research nature it cannot be used in a production environment. This shows the main difference between an industrial and reference codec. An industrial codec is optimized for a specific target platform or desired application.

When this limitation became evident the focus switched from creating an application to creating a proof–of–concept demonstration. This demo shows the possibilities of the original proposal, but the video is pre–encoded and decoded in advance. To create this demonstration as accurate as possible, a model was created and measurements where done on the JSVM codec.

Next to the demonstration, a separate part was added to the project. This was the improvement of choosing the GOP size adaptively in JSVM. This is less related to the previously sketched context, but still related to the research topic AVC.

1.5 Document structure

Chapter 2 introduces the technologies used in video conferencing, it discusses different standards and network technologies involved. A large focus of the project was on the technologies behind video coding. The architected application uses scalable video coding which is explained in chapter 3. The use of scalable video in a next–generation video conferencing architecture, together with the introduced technologies, is discussed in chapter 4. Because the reference implementation of JSVM does not allow the creation of an application, a demonstration program was made as described in chapter 5. The proposal of an improvement of the algorithm for selecting GOP sizes adaptively is discussed in chapter 6. Chapter 7 concludes this document, with conclusions and recommendations regarding the project and work done.

Chapter 2

Multipoint video conferencing

> *multipoint video conferencing:* "A discussion between three or more (groups of) people who are in different places but can see and hear each other using electronic communications."

This chapter gives an introduction of video conferencing in general, with a focus on multipoint video conferencing. Section 2.1 introduces the setup of a video conference and some commonly used terms in video conferencing. Section 2.2 highlights the media transported during a video conference. The way this media is transported over the network is explained in section 2.3. As there is a focus on improving the perceived quality, this is discussed in section 2.4.

2.1 Creating a conference

2.1.1 Generic steps

A video conference is a communication session helt between different kinds of terminals, also called the endpoints of a network. In network terminology, a video conference is a type of communication session. For having such a communication session, the following generic steps have to be taken[1]:

1. Establish connection
   First the different endpoints must connect to each other. They must know where to send the data, what kind of data will be communicated (audio, video or both) and what protocol is used to transmit the data. This is established in the first step.

2. Communicate data
   When a connection is made, data can be transferred between the endpoints. In a multipoint video conference this data is the audio/video that is transmitted between the endpoints. Beside audio/video, additional control signals can be sent. These signals can be used to alter the parameters of a conference, such as the number of speakers. The control signals can also be used to provide feedback on the reception quality of the received data.

3. Terminate connection
   When an endpoint quits a communication session, the connection must be terminated. The other endpoints must be notified before quitting. Especially when multiple endpoints are connected, this is an important step.

[1] These steps are often split into smaller sub–steps.

2.1.2 The standards H.323 and SIP

For establishing and maintaining a communication session, two major standards exist: H.323 and the Session Initiation Protocol (SIP). H.323 is a recommendation from the International Telecommunication Union (ITU) that defines an entire suite of protocols to provide different kinds of communication sessions on a packet oriented network, such as the Internet. Because H.323 covers a large range for communicating in all sorts of environments, it is often called an 'umbrella standard' [13, 29].

SIP is a standard defined by the Internet Engineering Task Force (IETF). As its name implies, it is purely meant for initiating, modifying, and terminating a communication session that involves multimedia such as video, voice or instant messaging. Unlike H.323, SIP does not define the protocols to use for transporting the media [25, 30].

An elaborate review and comparison of these standards will not be given, as the focus of the project is on the processing on a set–top box and it is assumed that the network is not a bottleneck. One notion worth mentioning is that SIP is often stated to be more flexible and scalable than H.323, thus seams preferable in a new video conferencing application.[24, 20]

2.2 Transported media

During a video conference session, media is transmitted between endpoints in the form of audio/video data, as explained in section 2.1. To save available bandwidth of the network this media is compressed using a codec (**co**mpressor / **dec**ompressor). For compressing audio and video an entire range of codecs is available, such as GIPS, GSM, PCM or G.7x for audio and H.26x, MPEG-2 or MPEG-4 for video. Compression of the media data creates a coded bit–stream that can be transported.

To limit the work to be done it was decided that audio would not be used in the proposed application. Secondly, it was already decided that for compressing video AVC would be used. During the technology study, scalable video coding was encountered. This is more elaborately discussed in chapter 3, the use of scalable video in a multipoint video conference is explained in chapter 4.

2.3 Media transport

The compressed media mentioned in the previous section has to be transported over the network. In 2.3.1 options to connect multiple endpoints are discussed. The ways to transmit media between multiple endpoints are discussed in 2.3.2. 2.3.3 explains the use of RTP for packetizing and transporting a bit–stream.

2.3.1 Connection methods

In multipoint video conferencing there are two solutions for connecting several endpoints: centralized and decentralized. Both methods are depicted in figure 2.1. In the centralized method, the endpoints send their media to a central server. This server combines all these streams to one stream and sends this to the different endpoints. In the decentralized method the task of combining multiple streams is placed at the endpoints.

A central server has several drawbacks: First, a central server represents a single point of failure. Besides that, it can cause long delays in conferences where participants are located far from each other. Because of this it was decided to use the decentralized method,

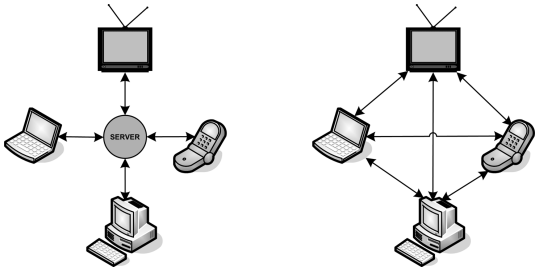


Figure 2.1: Centralized versus decentralized connection

also because it could be used together with multicasting (as explained in the next subsection).

2.3.2 Transmission methods

When multiple endpoints in a network have to communicate with each other, three different methods of transmitting data can be identified: unicast, broadcast and multicast. The difference between these methods is shown in figure 2.2, where endpoint 1 sends an audio and video stream to the other endpoints.

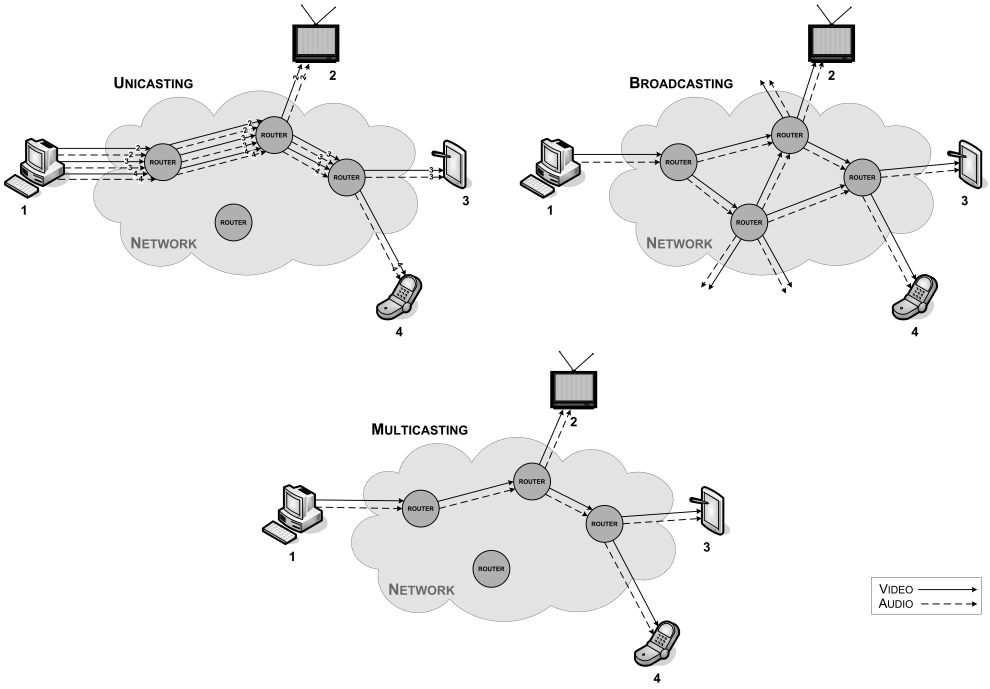


Figure 2.2: Unicasting, broadcasting and multicasting

Using unicasting, for each stream a separate connection has to be made. When multiple endpoints are connected, this results in several point–to–point connections. The downside of this method is that, if a large group of endpoints has to communicate, the same media streams are sent multiple times over the same network link. This can flood the upstream channel from an endpoint to a router and the channels from router to router, a waste of bandwidth. A solution of this problem can be the use of a central server, but as

stated before this is not recommended.

An alternative is broadcasting, where the streams are sent to every endpoint in the network. A receiving endpoint decides if it is interested in the stream. The downside of this method is that the entire network is flooded, while only a small portion of endpoints is interested in the actual broadcast. Just like unicasting, this is a waste of bandwidth.

A solution to this problem is multicasting, which is a method for efficiently sending information from one to multiple receivers at the same time. This is done using a multicast address to which different users can 'tune in'. If data is sent to this address, only the endpoints tuned in receive the information. In figure 2.2 this is shown as two streams sent over the network. The endpoints can choose to receive one or two streams, depending on the network and processing capabilities. Thus, from a network perspective, multicasting is an efficient way to distribute media between multiple endpoints. As will be explained in chapter 4, multicasting can also be combined with scalable video coding.

Although the concept of multicasting dates from 1989 [7], it is still not widely available. The support for multicasting is still limited to separate Local Area Networks (LANs), because it requires more complex network components, such as routers, switches and firewalls. Currently multicasting is slowly moving from the LAN area to the Internet.

2.3.3 The Real–time Transport Protocol

The transmission of data through an IP network is done packet based. A media stream is divided into smaller packets which are transmitted separately over the network. This transmission is done using a protocol stack, each layer of the stack defines a specific protocol needed for data to be transported [27, 10]. In general, the lowest layers are for the physical and link connection. The next layer (mostly the Internet Protocol (IP) layer) provides a network link between several endpoints, using IP addresses. On the Internet, on top of IP two core transport protocols are used for sending data: the Transmission Control Protocol (TCP) and User Datagram Protocol (UDP).

A video conferencing application is a typical real–time application, that places specific requirements on the transport protocol. In a real–time application, the receiver of a media stream has to play this stream at the moment it is received, instead of buffering the entire stream. For this reason, the timing of the arrival of packets should be predictable, but some variation can be dealt with using a small buffer at the receiver. A lesser requirement is the reliable delivery of all packets by the network. Reliable delivery is desirable, but many audio and video applications can tolerate some loss.

TCP is a connection oriented protocol, focused on reliable delivery of packets. The downside of this approach is that it can cause unpredictable delays in the delivery time, an unwanted feature for real–time delivery. UDP on the other hand is a connectionless protocol, based on best effort delivery. It supports the casting methods described in section 2.3.2. The problem with UDP is that it basically only provides the use of a source and destination address/port. It does not provide any reliability to the transport, nor can it affect the timing of packet delivery [8, 19].

To solve the limitations of TCP and UDP the IETF has developed the Real-time Transport Protocol (RTP) together with the Real-Time Control Protocol (RTCP). The place of RTP in a protocol stack is on top of a transport protocol. In most applications it is located on top of UDP, as shown in figure 2.3.

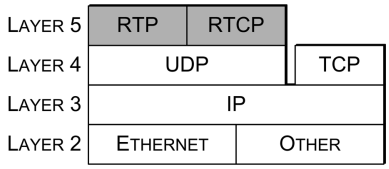


Figure 2.3: RTP protocol stack

The RTP protocol extends the underlying protocols with functions useful for real–time media transport. These functions include *timing recovery*, *loss detection*, *payload identification* and *source identification*. The accompanying protocol RTCP can be used to synchronize multiple RTP streams, by sending a reference clock or it can be used to provide reception quality feedback on an RTP connection. For the aimed video conferencing application, especially the payload identification function of RTP is interesting. It provides means to identify the content of a packet at the time of reception. This can be used together with scalable video coding to filter out packets, as will be explained in chapter 4.

2.4 Perceived quality

2.4.1 Quality of a conference

The quality of a multipoint video conference depends on more than just the video quality. As already mentioned, to create a higher perceived quality, the active speaker should be better visible. The active speaker should 'stand out' in order to obtain the user focus. The proposed architecture uses scalable video that allows flexibility in changing the video quality. An exact definition of what creates a higher perceived quality was not specified, the application is capable of adjusting parameters for achieving this. A survey on the quality of an H.323 video conference, from the Ohio State University, gives an indication for prioritizing quality parameters [5]:

- Smooth (continuous) video is more important than picture quality. The variation in delay, the jitter, should be as minimal as possible. Within the context of the performed survey, a jitter larger than 50 ms is observed as poor quality.
- Picture quality is more important than delay. Quite surprising, a (small) delay in conferencing is allowed, as long as it improves the picture quality. Obviously audio and video should remain in sync. In [5] it was observed that a delay up to 300 ms is acceptable.

2.4.2 Measuring quality

For evaluating or comparing the quality of a video system, two different kinds of measurements are possible: objective and subjective. The objective method compares an in- and output of a video compressor by calculating the mathematical difference. The most used calculation is the peak signal-to-noise ratio (PSNR), as defined by equation 2.2. It uses the mean-square-error (MSE) as defined in 2.1. Here X and X' represent an input and output picture of the video, p and q are the amount of horizontal and vertical pixels of a picture. This is further explained in chapter 3.

$$MSE(X, X') = \frac{1}{mn} \sum_{p=1}^{m} \sum_{q=1}^{n} \left(X[p,q] - X'[p,q] \right)^2 \tag{2.1}$$

$$PSNR(X, X') = 20 \log_{10} \left\{ \frac{255}{\sqrt{MSE(X, X')}} \right\} \tag{2.2}$$

Typical PSNR values range between 20 and 40 dB and are usually reported to two decimal points (e.g. 25, 47). The larger the PSNR the better. Next to the PSNR, other calculations exist such as: Structural SIMilarity (SSIM) and Video Quality Metric (VQM). The advantage of an objective measurement is that it gives precise results that can be reproduced. The downside is that none of the objective methods can represent the subjective experience of a human observer.

The perceived quality of a video system is based on a lot of different factors, for instance the task of a video system. Is a user passively watching a movie, or is he actively involved in a video conference? To be able to measure these factors, subjective tests can be performed. These are user tests, where video sequences are shown to a group of viewers. Their opinion on the quality is averaged to evaluate the quality of a video system. Different kinds of user tests are standardized in the ITU-T recommendation BT.500-11 [14].

2.5 Conclusion

Technologies involved with multipoint video conferencing are introduced, such as the standards H.323 and SIP and the connection/transmission methods. Protocols used for establishing/maintaining a conference are not studied in this thesis. Within the context of a fully connected world it is preferable to use a decentralized method for combining several media streams. Multicasting provides an efficient use of the available bandwidth. The presented video conferencing architecture can use multicasting combined with scalable video and the RTP protocol. This is discussed in chapter 4.

Measuring the quality of a video system is discussed. When developing a video system, objective tests are an efficient way for measuring the video quality, but they cannot reproduce the subjective experience of a user. To measure this experience, it is recommended to perform subjective tests.

Chapter 3

Scalable video coding

> *vid·e·o (n.pl):* "The technology of capturing, recording, processing, transmitting, and reconstructing moving pictures, typically using celluloid film, electronic signals, or digital media."

This chapter gives an introduction to video coding in general and scalable video in particular. Section 3.1 introduces the components of a video encoder and the concept of scalable video. The techniques used in a video encoder are explained in section 3.2. The AVC standard and its scalable extension JSVM are discussed in 3.3.

3.1 Video coding concepts

3.1.1 Generic video coding

The goal of video coding is to compress a video signal, while preserving as much of the original signal as possible. If the original quality is fully preserved the compression is called *lossless*. When de decoded video differs from the original it is called *lossy*. A video signal consists of a sequence of frames and the amount of frames per second determine the *temporal resolution* or framerate of a video. A single frame consists of $m \times n$ pixels that represent the color/brightness of a point in a frame. The amount of pixels in a frame define the *spatial resolution*.

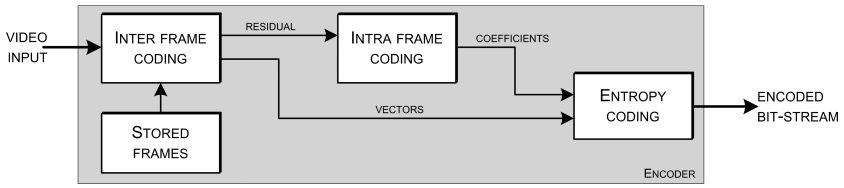


Figure 3.1: Generic video coding concept

To produce an encoded bit–stream, a video encoder has three functional components as shown in figure 3.1. Each component exploits redundancy within a video sequence. *Inter frame coding* removes temporal redundancy by exploiting similarities between neighboring frames. This component uses a buffer of previously encoded frames. The similarities between neighboring pixels in a frame is used to remove spatial redundancy. This is called *intra frame coding*. To produce a compressed output stream, *entropy coding* is used. The entropy coder removes statistical redundancy in the output of the previous two components. The techniques used in the different components are discussed in section 3.2.

3.1.2 Scalable video coding

Normally, the decoding of an encoded bit–stream results in a video sequence with the same spatial and temporal resolution as the original input video. It is also possible to create a bit–stream from which a video at other levels can be extracted. Such a bit–stream consists of a base layer that represents the lowest quality possible and several enhancement layers. Each enhancement layer adds 'quality' to the base layer and if every enhancement layer is decoded the highest possible quality is achieved. An encoded bit–stream can have three dimensions of scalability as shown in figure 3.2: spatial, temporal and quality scalability.

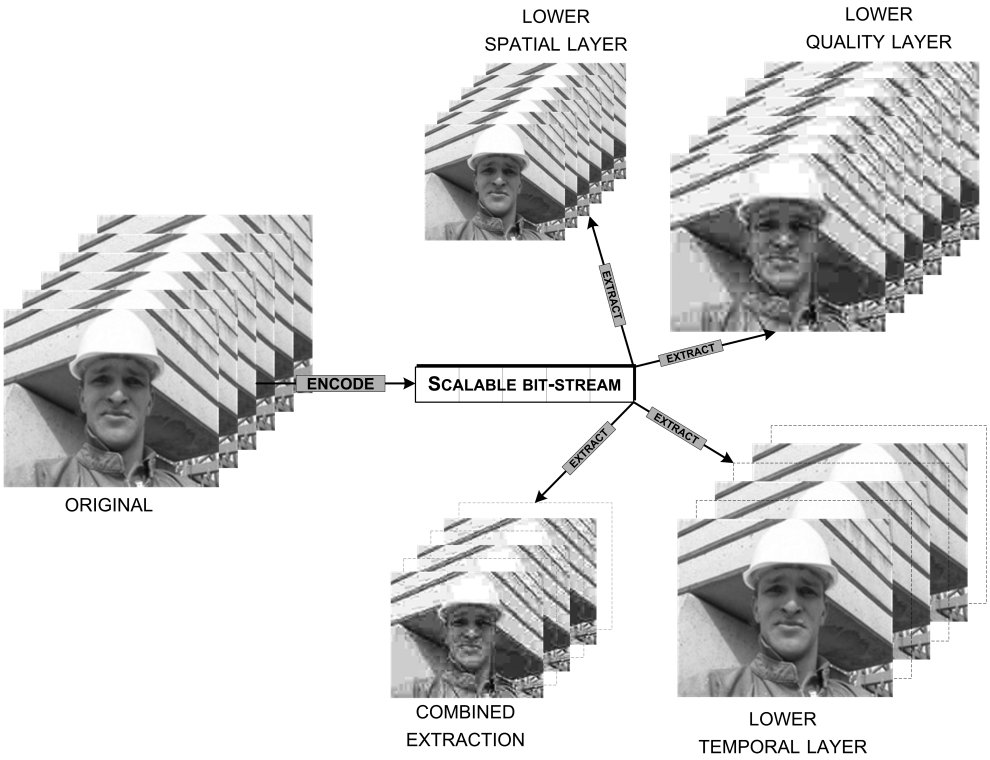


Figure 3.2: Different dimensions of scalability

Spatial scalability refers to scaling in the amount of available pixels, temporal scalability to scaling in framerate and quality scalability to scaling in bitrate. Scaling in bitrate is done at the cost of more blocking artifacts in a frame. This is achieved by scaling the quantization factors used in intra coding, as explained in 3.2.2. The different dimensions of scalability can be combined into one bit–stream as shown in figure 3.2. This figure shows a scalable bit–stream from which a video at a lower framerate, resolution or quality can be extracted. Of course, these dimensions can also be combined.

The visual result of spatial scalability can be the same as quality scalability, because less available pixels also result in a more 'blockish' picture. Because of this, spatial scalability is also referred to as course–grain scalability and scaling in bitrate as fine–grain scalability.

3.2 Functional coding components

3.2.1 Inter frame coding

The goal of the inter frame coding is to remove similarities between subsequent frames. This is done using a block based method which uses *motion estimation* and *motion compensation*.

Motion estimation

A frame of $n \times m$ pixels is divided into macroblocks of $x \times y$ pixels, normally of size 16×16. Motion estimation on a frame involves searching a macroblock in a reference frame that as closely as possible matches the current macroblock to be encoded. A reference frame is a previously encoded and stored frame from the video sequence. Because frames can be encoded in a different order than they arrive, a reference frame can be before or after the current frame in the original input order. The original order is called the display order. The result of motion estimation is a motion vector, that describes the position of the reference macroblock relative to the original macroblock. The concept of finding a macroblock during motion estimation is shown in figure 3.3.

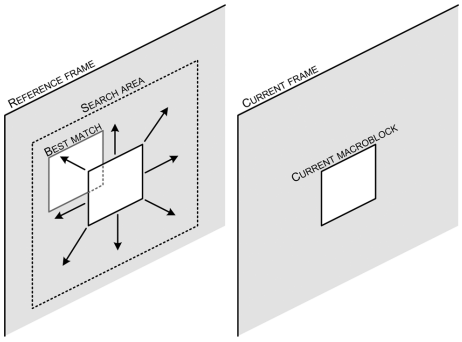


Figure 3.3: Motion estimation

Motion compensation

Because the found reference block almost never represents the original block entirely, motion compensation is needed. In motion compensation the found *reference macroblock* is subtracted from the *original macroblock*. This results in a *residual macroblock* that is further encoded using intra coding. Within the encoder, the residual macroblock is decoded again and added to the reference block. This results in a *reconstructed macroblock*. All the reconstructed macroblocks of a frame create a reference frame that can be used for future predictions. Instead of using an original frame as reference, the en- and decoding of such a frame ensures that the encoder and decoder use an identical reference frame for motion estimation.

Frame types

For motion estimation different kinds of predictions are possible: none, forward and bidirectional. With respect to these prediction types, different frames can be identified. Generally there are three types of frames: I–, P– and B–frames[1]. On I–frames, short for *intra* frames, the motion estimation and compensation steps are omitted. These frames

[1] The terminology for these frames can differs per standard.

are necessary at the start of a sequence and to allow random access of a video stream. The P– and B–frames, short for *predicted* and *bidirectional* predicted frames, are inter coded frames. The difference between the two is that P–frames are only predicted from previous frames in display order whereas B–frames can be predicted from previous and/or future frames.

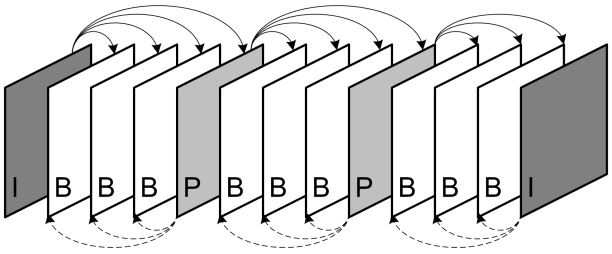


Figure 3.4: I–, P– and B–frames

The encoding of a video sequence most often is done using a Group of Pictures (GOP). The boundary of such a GOP is determined by an I–frame. A typical ordering in a GOP used in MPEG–2 is shown in figure 3.4. The dashed lines show the backward predictions.[2] Note that in order to encode the sequence shown in figure 3.4, the frames have to be encoded in the following order: IPBBBPBBBIBBB. This shows the difference between the display and encoding order of a video sequence.

3.2.2 Intra frame coding

In the previous discussed inter frame coding component, a prediction is used to remove redundancy between frames. This concept can also be used within a single frame, as neighboring pixels are also related. In general, to remove spatial redundancy a transformation and a quantization of the results of this transform is applied. A transformation converts the input samples to another domain where they are represented by different transform coefficients. These coefficients are quantized to remove insignificant values and leaving a small number of coefficients that represent the input frame. Quantizing the transform coefficients is the step where information is removed from the original signal. If no quantizing is applied, the compression is lossy. In quality scalability the coarseness of the quantization of the coefficients is scaled. Besides transforming and quantizing, in some encoders, such as AVC, it is also possible to use predictions on a pixel level, where a pixel is predicted from surrounding pixels.

3.2.3 Entropy coding

The entropy coder compresses the output from the inter coding step (the motion vectors) and the intra coding step (quantized coefficients). Entropy coding consists of removing statistical redundancy of the input data. For example, by representing an often reoccurring motion vector by a small binary code. Entropy coding results in a compressed bit–stream that consists of coded motion vectors, coded residual coefficients and additional header information. This bit–stream can be transmitted or stored.

[2] The picture shows an open GOP, a closed GOP ends on a P–frame. (Backward prediction from the second I-frame is omitted.)

3.2.4 The block based hybrid model

The combination of inter, intra and entropy coding is usually known as a *block based hybrid video compressor*. Figure 3.5 depicts a schematic overview of such a codec. Almost every video compressor, such as MPEG–1,2,4 or VC1, uses this scheme. Also the AVC codec is based on this concept. To reconstruct the video sequence from a coded bit–stream, a decoder has to apply inverse quantization, inverse transformation and motion compensation.

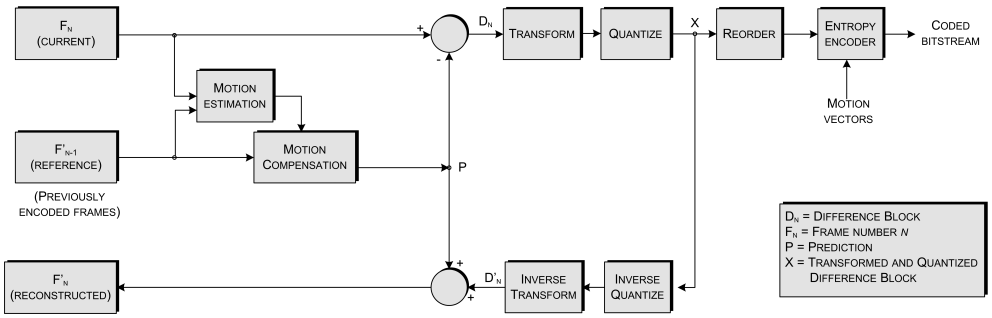


Figure 3.5: Block based video encoder

Note that the previous subsections only highlight the concept behind the different encoding components. The techniques used, determine the performance of a video encoder. The discussion of the different techniques is beyond the scope of this document.

3.3 The H.264/AVC standard

3.3.1 Background

AVC is one of the newest video coding standards developed by the ITU-T Video Coding Experts Group (VCEG) and the ISO/IEC Motion Pictures Expert Group (MPEG). VCEG and MPEG formed the Joint Video Team (JVT) in order to jointly standardize the codec. Officially the standard is entitled Advanced Video Codec (AVC) and was published in 2003 as MPEG–4 Part 10 and ITU-T recommendation H.264. In this document the codec is referred to as AVC. AVC has been adopted by major industries for transmitting or storing video. These include those present the Digital Video Broadcast (DVB) consortium, the Blu–ray group and the 3G Mobile Group.

3.3.2 Overview

Figure 3.6 shows a block diagram of a typical AVC encoder. As shown, it resembles the generic block based encoder, with an addition for intra frame prediction. The AVC standard is an evolution of previous video compression standards, with minor improvements in all the different components. Together, these small improvements deliver a coding efficiency that is roughly twice as good as MPEG–2, but at the cost of a higher computational complexity. It is beyond the scope of this document to discuss every improvement of AVC. The interested reader can refer to [31, 23] for more information.

One notable difference compared to MPEG–2 is that B–frames can be used as a reference frame for other frames. This feature is used to create temporal scalability in the JSVM codec as discussed in subsection 3.3.4. Another important concept that is used in the

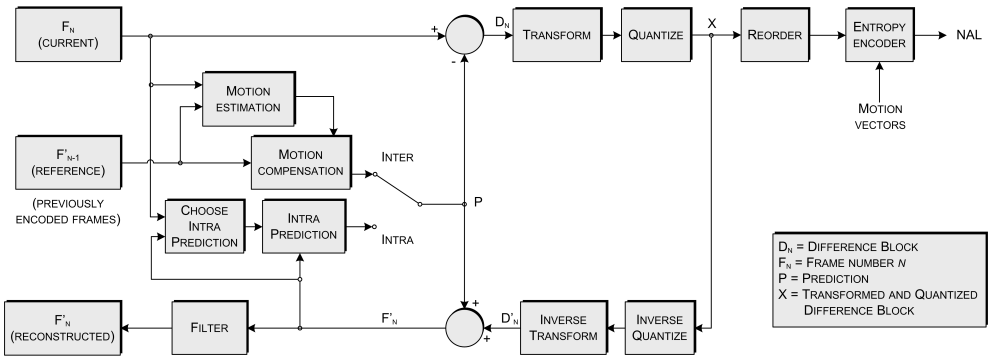


Figure 3.6: AVC video encoder

video conferencing architecture is explained next, the Network Abstraction Layer (NAL) and NAL units (NALUs).

3.3.3 Network abstraction layer

In AVC the coding of the video is split in a Video Coding Layer (VCL) and a Network Abstraction Layer (NAL). The Network Abstraction Layer extends a traditional video codec (that focusses on the creation of a bit–stream) with an additional packaging scheme. This provides the ability to map the coded bit–stream, the Video Coding Layer (VCL) data, to different transport media such as RTP/IP, H.32x or MPEG transport streams and different file formats such as MP4 or MMS.

An encoded bit–stream is organized in several NAL units (NALUs). There are different type of NALUs for motion vectors, coded blocks and sequence/picture parameters sets. Basically a NALU is a packet including a header that identifies the type of data inside the packet. The NALU structure specifies a generic format suitable for both packet-oriented and bit–stream-oriented transport systems. For example, one or more NAL units (NALUs) can be put into a RTP packet. The header information of a NALU allows for early detection of the contents and unequal error protection, where packets with more important information are better protected against transmission errors [2].

3.3.4 The Joint Scalable Video Model

Background

The Joint Scalable Video Model (JSVM) is an extension to AVC that provides the three dimensions of scalability mentioned in 3.1.2. JSVM is currently under development by JVT and technical maturity is expected around mid 2006 [1, 28, 21, 22]. A broad review on how JSVM achieves the different types of scalability is beyond the scope of this document. Here the structure of the bit–stream is discussed and one dimension of scalability, temporal scalability, is explained as it is related to the part discussed in chapter 6 (adaptive GOP). An example of a typical JSVM encoder that provides combined scalability is explained in appendix A.

Bit–stream structure

Figure 3.7 shows the formatting of a scalable bit–stream from which the different quality layers can be extracted. Here the original order compared to the encoding order of the frames is clearly visible in the bit–stream. In JSVM additional NAL units (NALUs) are

added to AVC to allow identification of the different quality layers. The coded information of one frame is split into several NALUs that allow easy identification of the different spatial layers. Quality (fine–grain) scalability is more flexible: The quantization information is placed in so called progressive refinement slices that can be cropped at an arbitrary point when a desired bitrate/quality is reached [1, 31]. Note that for extracting a desired layer, all the preceding NALUs have to be decoded.

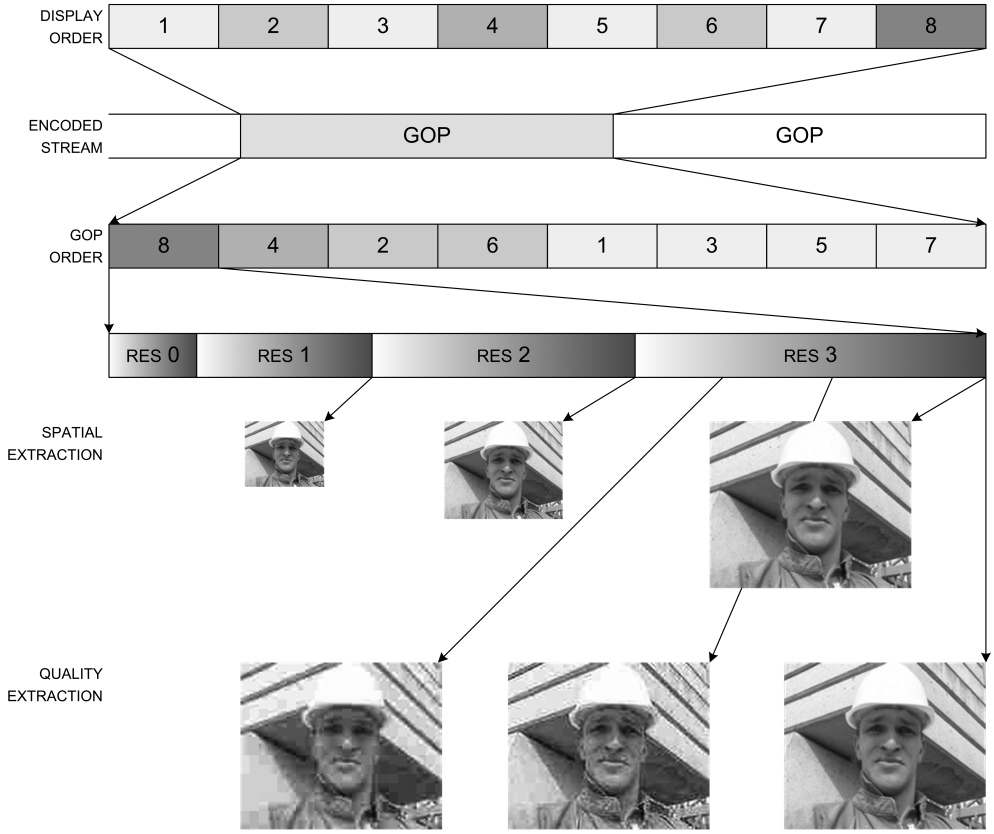


Figure 3.7: JSVM scalable bit–stream

Temporal scalability

Temporal scalability within JSVM is based on the concept of hierarchical B–pictures[3]. This is a specific prediction structure as shown in figure 3.8. The structure shown results in a stream with four temporal levels L_0 to L_3.

The lowest level, representing the lowest framerate, is encoded using I– or P–frame encoding that can be identical to the techniques used in AVC. This ensures that the base layer is decodable by any AVC decoder. The B–frames in between are predicted hierarchically as shown. This results in an encoded stream where frames of a specific temporal layer do not depend on frames from a higher layer. Thus, frames of a higher temporal layer do not have to be decoded to extract a lower temporal resolution.

Note that because of the hierarchical prediction structure, the encoding order of the frames in this example is: 0 8 4 2 6 1 3 5 7... This means that, when decoding the

[3] For a long time during development Motion Compensated Temporal Filtering (MCTF) was used, but is was discovered that the benefit of MCTF compared to hierarchical B–pictures is limited [22].

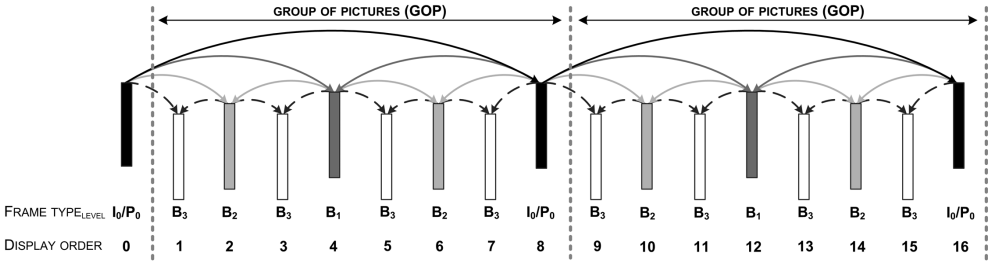


Figure 3.8: Hierarchical B–pictures

full stream, there is a structural delay of four frames. At the encoder, first a full GOP has to be read. Or more general, if there are N temporal enhancement layers, the structural decoding delay is 2^{N-1} frames. This structural delay has an impact on the required buffering of an en- or decoder. A second notion is that N temporal enhancement layers require a GOP with a size of at least 2^N, but a larger GOP size may improve the coding efficiency. This is exploited by using a variable GOP size, as explained in chapter 6.

3.4 Conclusion

An introduction was given of video coding in general and the concept of scalable video coding was explained. The different components of a video encoder were highlighted. The discussed concepts behind inter, intra and entropy coding show just an overview of video encoding. For each encoding component, different techniques exist. The AVC standard was introduced and its relation with JSVM. The easy identification and extraction of quality layers possible with JSVM, is used in the proposed multipoint video conferencing architecture discussed in the next chapter.

Chapter 4

Architected system

> *Software architecture:* "The structure or structures of a system, which comprise software elements, the externally visible properties of those elements, and the relationships among them." [3]

This chapter discusses alternatives of an architecture for a next–generation video conferencing system. The architecture is aimed at improving the perceived quality, while making efficient use of the resources of a platform and network. The approach is to use scalable video coding combined with multicasting. First an overview is given of a generic video conferencing endpoint in section 4.1. The concept of combining multicasting with scalable video is explained in section 4.2. Section 4.3 discusses the design of an architecture of an endpoint within this concept.

4.1 A video conferencing endpoint

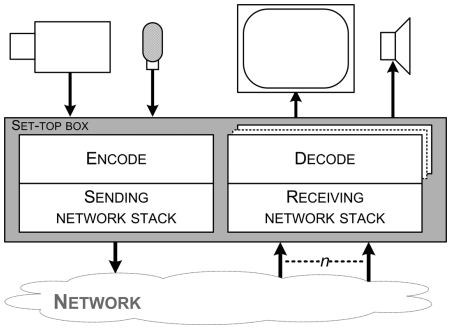


Figure 4.1: A generic video conferencing endpoint connected to n other participants

An overview of a video conferencing endpoint connected to n other participants is shown in figure 4.1. Such an endpoint consists of a sending and a receiving side. The sending side uses a camera and microphone to capture audio and video which is compressed using encoders. The outgoing network stack sends this information to the other n participants using one of the connection/transmission methods as described in 2.3. On the receiving side, the network stack collects the data from every participant. For each participant an audio and video decoder is needed to reconstruct the data, after which it is sent to the display and speaker[1].

[1] Some extra building blocks are needed for a full application, such as a component that combines several audio/video streams to a one stream and a component for handling user input. For ease of presentation these are not depicted.

4.2 Combining multicasting with scalable video

The use of scalable video has the advantage that a receiver has a choice in the quality of the video that is to be decoded. This allows heterogenous terminals to be connected, because a terminal can choose a quality level that optimally fits its resources. Secondly, the choice in quality also allows highlighting of the active speaker. By degrading the quality of the non–speakers, resources are freed that can be used for the active speaker. Enabling an endpoint to exploit this choice of quality, can be done using multiple multicast channels and by filtering at an endpoint.

The combination of multicasting with scalable video uses multiple channels instead of sending a scalable bit–stream using one channel. A bit–stream is split into several sub–streams, containing one or more scalability layers. These sub–streams are sent through a separate multicast channel. A schematic representation of mapping a video stream that consists of m different layers to n multicast channels is shown in figure 4.2.

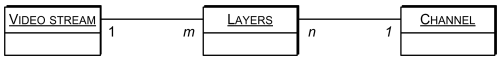


Figure 4.2: Mapping a scalable stream to several multicast channels

A receiver can choose the quality of the received video by tuning in to one or more multicast channels. This use of multicasting makes efficient use of available network resources. An example of this concept is shown in figure 4.3.

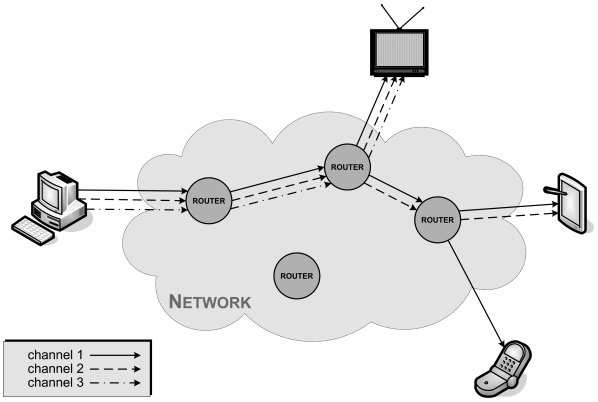


Figure 4.3: Using multiple multicast channels to send video

The different quality layers, residing in a (sub–)stream can also be extracted by applying filtering at a terminal. This filtering enables quality selection in an environment, where multicasting is unavailable.

The proposed concept has several potential advantages:

1. Heterogenous endpoints can be connected.
   In the fully connected world different kinds of terminals are connected, each with different features and capabilities. Because a choice in quality is available, all these different endpoints can have a video conference while still maintaining the maximum amount of video quality that is possible at an endpoint.

2. Efficient use of resources, while showing the active speaker at a higher quality.
   When it is necessary to show multiple videos on a screen, using scalable video coding it becomes possible to lower the quality of several streams, for instance

the non–speakers. This frees resources for the stream that is most important, the active speaker. This is an efficient way for highlighting the active speaker.

3. The base layer can be AVC compatible.
   The architecture was designed to operate only with endpoints that have a JSVM en- and decoder. But the encoder is capable of creating a base layer that is AVC compatible and the decoder can decode normal AVC streams. This allows compatibility with (older) endpoints that do support AVC but not the scalable extension JSVM. If this feature is required, the different quality layers should be carefully chosen.

4. Only open/standard techniques are used.
   This is relevant as the context of the video conference is at home, using a television. This means that different home users with different telecom/access providers should be able to communicate with each other. If some strange or expensive technique is used it is less likely that a provider or endpoint manufacturer is going to support it. The most specialized techniques used are adjustments in the network layer to be able to choose a quality level. Another requirement is that a provider should support multicasting.

4.3 Mapping a scalable stream to multicast channels

As described in chapter 3, JSVM has several dimensions of scalability. The NALUs containing the different layers can easily be mapped to different multicast channels. One way is splitting the stream and sending the NALUs of every different layer through a separate channel. The opposite of this solution is sending the entire stream through one channel, but apply filtering at the endpoint to extract a quality level. The architectures for these options are explained in sections 4.3.1 and 4.3.2. In both architectures it was assumed that a method exists for identifying the active speaker. (e.g., by analyzing the audio) Section 4.3.3 provides a review of these architectures.

4.3.1 A dedicated channel for every layer

An overview of an endpoint, sending a video stream consisting of m layers through m multicasting channels, is shown in figure 4.4.

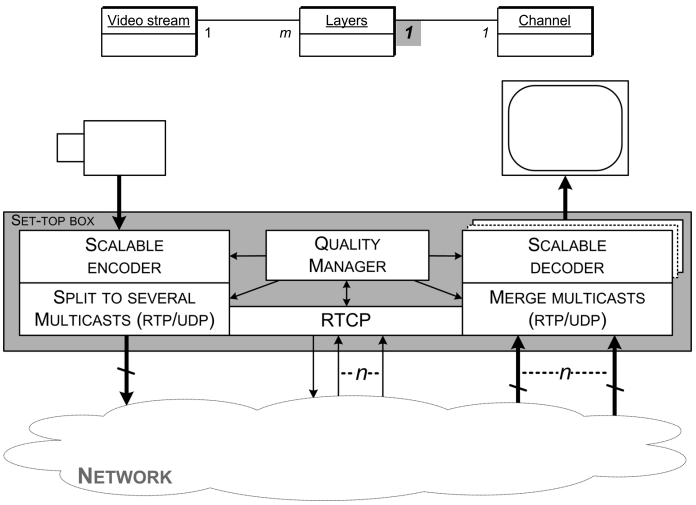


Figure 4.4: Architecture for mapping video layers to multicast channels

The microphone and speaker are not depicted, because audio was not studied during the project. For en- and decoding scalable video, the JSVM codec is used. The most interesting building blocks in this architecture are the multicasting components and the quality manager. The sending multicasting component splits an encoded bitstream into separate streams, containing the base and enhancement layers. The streams are sent to different multicast addresses using RTP/UDP. (A video stream from one user, consisting of several multicasted layers, is denoted with a crossed arrow: $\not\rightarrow$). The receiving multicasting component is tuned in to the different channels of connected participants. For each connected participant it combines and synchronizes packets to one bitstream that can be decoded. The synchronization of the packets of one participant is done using a RTCP channel.[2]

The quality manager is used to adjust the quality of shown videos. It keeps track of available platform resources and is responsible for changing the quality of the different connected participants. The quality manager instructs the receiving multicast component to how many channels to tune in. In order to highlight the active speaker, more layers can be decoded for this speaker than for the other participants. The quality at which the different videos can be shown is determined by available terminal resources, the number of participants to show and the required amount of resources for decoding the different streams at various quality levels. This is explained in more detail in chapter 5. Optionally the quality manager can be used to adjust the quality of the outgoing video streams, by changing encoding or transmission parameters.

4.3.2 One channel for entire stream

The opposite of the previous mentioned mapping is using one channel for sending the bit–stream. This architecture is discussed to show the concept of filtering. This architecture was designed for the use in an environment where multicasting is not available.

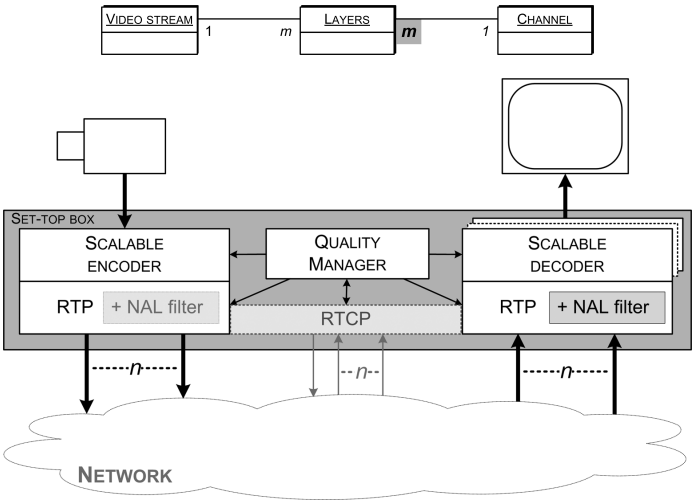


Figure 4.5: Architecture for filtering a scalable bitstream

This architecture shown in figure 4.5 still exploits the concept of using scalable video for adjusting the quality of the video at the receiving side. If multicasting is not available, this

[2] In future versions of the application, the RTCP channel can also be used for quality reception feedback. Using this feature, the quality of the video also can be adjusted at the sender, instead of only adjusting quality at a receiver. (e.g., if none of the receiving endpoints are capable of decoding a stream sent, the sender can lower the quality of his stream.)

architecture can be used by connecting the different endpoints using a unicast RTP/UDP connection. Instead of tuning into multiple channels, the determination of a quality level is done using a RTP component that uses payload identification for filtering NAL packets.

Because the video is sent using only one stream, a synchronization RTCP channel is not necessary anymore. Nevertheless it is still drawn, because when audio is added, it is still needed to synchronize the video and audio. Secondly, the drawn outgoing NAL filter is optional. This filter allows quality adjustment at a sender side that only affects a specific participant. (instead of changing encoding parameters, affecting every participant) As the network at LogicaCMG does not support multicasting, this architecture was chosen to create.

4.3.3 Review and recommendation

The architectures discussed, show two extremes for mapping quality layers. A problem with the first proposal is the amount of enhancement layers and channels needed. For example, in a conferencing application that uses three spatial and three temporal layers, the mapping of every layer to a separate channel results in nine streams and corresponding channels. These are only channels used for video, also an audio and a RTCP channel is needed. All these channels have to be multiplied by the amount of users connected. This large amount of channels can pose a problem at an endpoint, as the streams sent through these channels have to be synchronized. It also fills up the space on the network reserved for multicasting. This is problematic for an access provider, as it should be able to support multiple conferences.

A solution to this problem is a combination of both architectures. Although the second architecture was designed for a unicasting environment, filtering can also be applied on one multicast channel. Quality adjustment by tuning in to different multicast channels has the advantage that it does not require much processing of a receiver. But, due to the reason mentioned before, the amount of channels should be limited. In a combined architecture, a selection has to be made of what to map to a multicast channel and what to keep in one sub–stream. A feasible option is to use multicasting for different spatial layers and keep the other two layers in one stream. For the previous mentioned example with three spatial layers and three temporal layers, this results in three multicasted streams. Using this combination, an endpoint can connect to one or more multicast channels, depending on its screen size, processing power and network speed. For highlighting the active speaker or to adjust to processing fluctuations, filtering can be used.

4.4 Conclusion

Alternatives for a next–generation video conferencing application are discussed. The architectures allow an endpoint to choose the video quality using scalable video. This choice is used to adapt to network and processing resources of heterogenous terminals and to improve the quality of the active speaker. For adaptation of the video quality multiple multicasting channels and filtering can be used. Multicasting allows efficient transmission of the video and a choice in quality by tuning into one or more channels. The amount of channels should be carefully chosen. Filtering at an endpoint is a second option that provides quality selection. This can be used in environments where multicasting is unavailable. To limit the amount of channels, a combination of both methods is recommended.

Chapter 5

Researched system

> *Re·search (n):* "An active, diligent, and systematic process of inquiry aimed at establishing, interpreting and revising facts."

Before designing in detail a video conferencing application that uses filtering, some experiments where done on the JSVM codec. This was needed to become acquainted with the encoding parameters and the interface of the codec. During these experiments it was discovered that the codec was far to slow to en- or even decode a stream in real–time. (A sequence of ten seconds took several minutes to decode) As multiple streams were to be decoded simultaneously, together with the encoding of one stream, this posed a challenge to the project.[1]

To cope with this challenge it was decided to use pre–encoded and decoded videos. These videos are used to create a demonstration program that shows the difference between using scalable or non–scalable video on a resource constrained platform. The application also allows to make recommendations on the encoding parameters to use. Section 5.1 describes the model used for representing the decoding of videos at a different quality on a resource constrained platform. To use this model in the demonstration program, measurements were performed as discussed in section 5.2. The design and analysis of the demonstration program is in section 5.3.

5.1 Research model

Because pre–encoded and decoded videos are used, a model is needed to represent a decoding platform and to calculate what quality level can be achieved using this platform. The metric for creating this model is described in 5.1.1. To apply this model on a demonstration program, the quality of a reference video and the layering used in a video conference has to be defined. The choice of these quality levels is explained in 5.1.2.

5.1.1 Metric

The metric for creating the model is the *maximum computation time C* needed to decode a frame of a video sequence. Together with the framerate (or period T) of this video, this creates a budget needed to decode a specific video $B = <C, T>$. By defining a video stream that a platform should be able to decode in real–time, the corresponding decoding time creates a reference budget: $B_{ref} = <C_{ref}, T_{ref}>$. This budget represents the

[1] An important conclusion from this issue was already mentioned in chapter 1: A reference implementation of a codec cannot be used in a production environment.

maximum computation time available of a resource constrained platform.

Given the budgets needed for decoding different quality levels of a video, it is possible to calculate the highest quality possible. As long as the total amount of budget needed for decoding multiple streams does not exceed the reference budget, it is possible to decode them. Example (1) in figure 5.1 shows a platform with available budget <250, 1/60>. This platform is capable of decoding three identical streams that require <150, 1/30>. As it is possible with JSVM to extract a lower video quality that requires less computation, it is possible to free budget for the active speaker. This is shown in example (2), where two other quality layers are available with budgets <250, 1/30> and <150, 1/15>. Instead of decoding each video at <150, 1/30> it is possible to show the active speaker at a higher quality and even allow the decoding of a fourth stream.

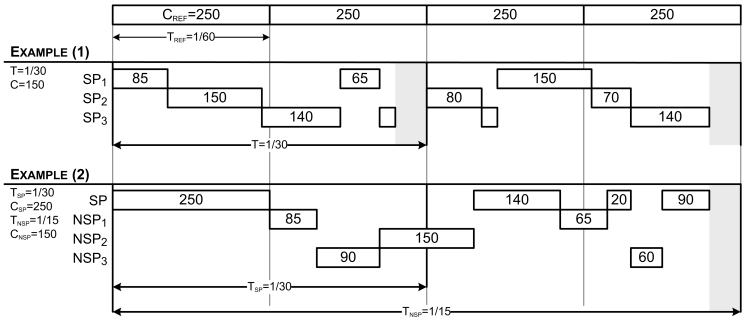


Figure 5.1: The decoding of three streams with reference budget <250, 1/60>

If multiple streams are decoded simultaneously, some overhead is created for switching between decoding the different streams. Therefore, the required budget for decoding multiple streams cannot be calculated by simply adding the individual budgets. To take this switching into account a second parameter is introduced, the multiplication factor $m > 1$. This factor represents the extra effort needed to decode more than one stream. *"The decoding of an additional stream requires m times its budget."* Figure 5.2 shows the addition of this factor m to the previous example.

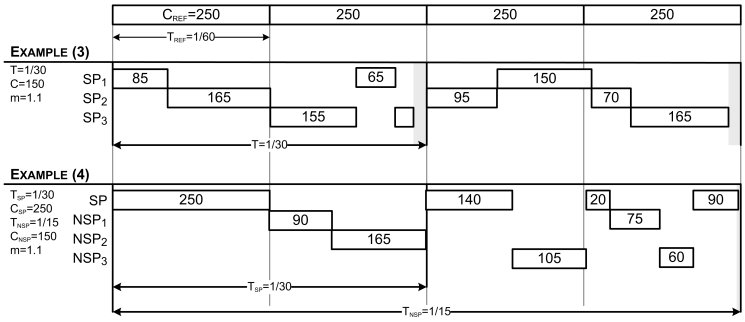


Figure 5.2: Decoding of three streams, including a multiplication factor

Note that, although the required budget increases, it is still possible to decode the streams. This is because in the first example, the reference budget is not fully utilized. Secondly note that the use of the *maximum* decoding time creates a worst–case scenario.

5.1.2 Video parameters

For using the previous sketched model in a demonstration application, the reference video and the used quality layers in a conference have to be defined. This is explained in this subsection. For denoting the characteristics of an encoded video the following notation is used: *size@framerate–bitrate*. For representing size, the Common Interchange Format (CIF) is used (as shown in table 5.1).

Name	Resolution
QCIF	176 × 144
CIF	352 × 288
4CIF	704 × 576

Table 5.1: Definition of QCIF,CIF and 4CIF

Reference video

The reference video is needed to define a reference budget. This video is based on the fact that a next generation set–top box should be capable of decoding a high quality AVC video in real–time. Three videos representing this quality level, called "Crew, Harbour and Soccer", are provided with the JSVM codec. These streams have a spatial resolution of 4CIF, a framerate of 60fps and a maximum bitrate of 3072 kbit/s. (4CIF@60–3072)

Conference video layers

For using scalability in a video conference, different quality layers were defined as shown in table 5.2. The determination of these layers was done by first defining a maximum and a minimum quality. The minimum is based on low quality, mobile video conferencing (QCIF@7.5–96) and the maximum is based on standard definition video quality (4CIF@30–2046). It was decided that for each type of scalability, three different layers

Resolution	Framerate [fps]	Bitrate [kbit/s]
QCIF	7.5	96
QCIF	15	192
CIF	7.5	192
CIF	7.5	288
CIF	7.5	384
CIF	15	256
CIF	15	384
CIF	15	512
CIF	30	384
CIF	30	576
CIF	30	768
4CIF	15	768
4CIF	15	1024
4CIF	15	1536
4CIF	30	1024
4CIF	30	1536
4CIF	30	2048

Table 5.2: Different quality layers for conference video

should be available. The bitrates in between are based on values from the website of HHI [1]. A step in bitrate within a spatio-temporal level[2] roughly improves the PSNR of a sequence with 2–3 dB. For creating these layers a new video was captured that represents a video conference. This video was encoded at the maximum quality and from this video the lower layers were extracted.

[2] the same framerate and resolution

5.2 Measurements

Two different measurements were done on the JSVM codec. The first was to determine the reference budget and the budget needed for decoding the different quality levels. This measurement is described in subsection 5.2.1. These measurements showed some unexpected behavior. To investigate this further, a second measurement was done as described in 5.2.2.

5.2.1 Decoding times

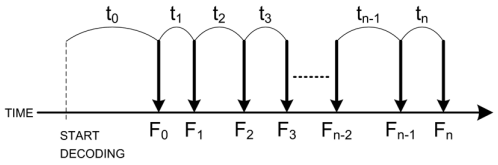


Figure 5.3: Measurement method

For determining the maximum decoding times, a timer was added to the JSVM decoder to measure each time interval between the output of two frames. From these measurements three different values were determined: The maximum decoding time of a frame, the average decoding time and the delay. The measurement method is depicted in figure 5.3. The timer is started when the decoding starts, when a frame written to the output the next timing interval is started. The three desired values can be determined using equations 5.1–5.3.

$$t_{delay} = t_0 \tag{5.1}$$

$$t_{average} = \frac{1}{n}\sum_{m=1}^{n} t_m \tag{5.2}$$

$$t_{maximum} = \max_{1\leq m\leq n}\{t_m\} \tag{5.3}$$

Tables B.1 and B.2 in appendix B show the results of this measurement. As is shown in B.2 the budget needed increases if a quality layer is added. A second important observation is that the different types of scalability have different impact on the decoding times. Within a spatio-temporal level, the change in bitrate, has a small impact on the needed budget, but scaling in spatial resolution has a large impact.

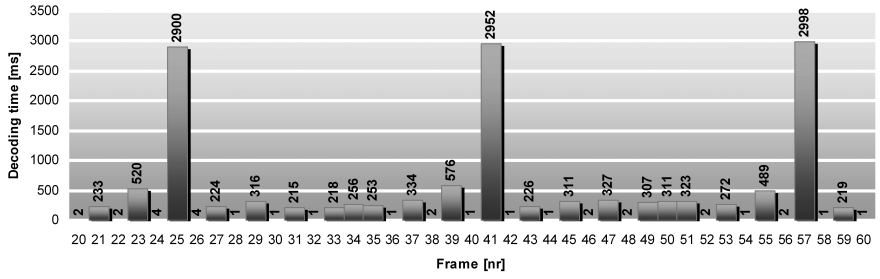


Figure 5.4: Example of jumps in decoding times

Another thing noted when analyzing the measurements was a large difference of the different decoding times within one sequence. This varied from the $t_{maximum}$ to one millisecond. An example of this anomaly is shown in figure 5.4, which shows a part of the

measurement of a reference stream. This unexpected behavior explains the large difference in measured budget between the different reference videos. As the difference between the reference videos is the GOP size, this resulted in the observations that the reordering of pictures creates a structural delay (as mentioned in chapter 3). At the encoder there is a delay of a full GOP, at the decoder a delay of half a GOP. To investigate this further a second measurement was done which is discussed next.

5.2.2 Bitstream structure

The second measurement consisted of analyzing the bitstream. The goal was to gain insight in the structure of the bitstream and relate this to the moment a frame was written to the output. In this measurement a log was kept of when a specific NAL unit was read by the decoder and when the corresponding frame was sent to the output. These measurements created extensive log–files, a graphical representation of part of such a log is shown in figure 5.5 and 5.6.

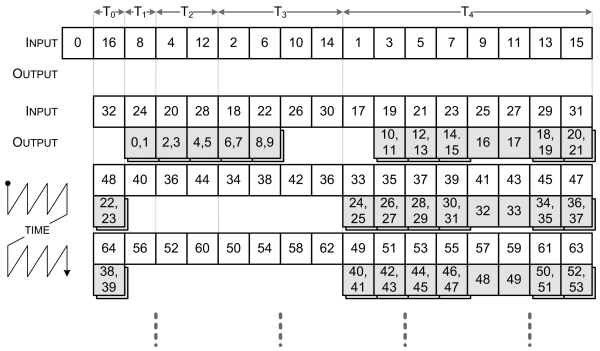


Figure 5.5: Measurement Crew: GOP = 16

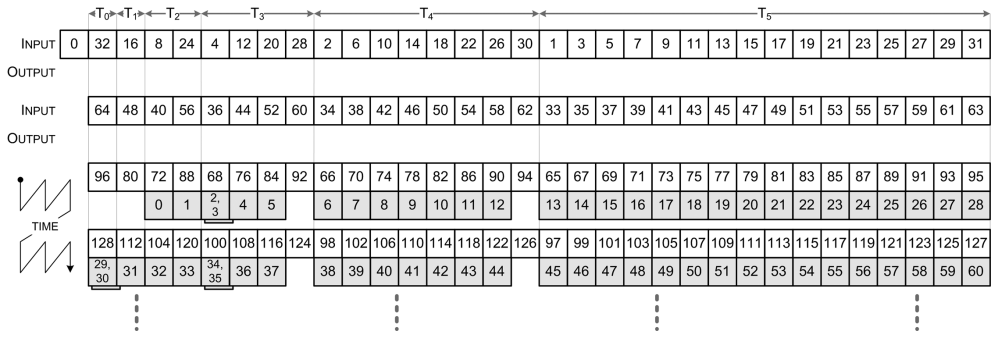


Figure 5.6: Measurement Soccer: GOP = 32

The first thing notable is that indeed the stream is formatted in reordered GOPs that create a structural delay. But the measured delay of the output of a frame is much higher than needed. For example in figure 5.5 the first frame is outputted when the NALs of 24 frames are read. This delay is higher than expected, because when half a GOP is read and decoded, there is enough information available to output the frame from the beginning of this GOP. It is expected that this is an implementation issue. As the codec is written to read/write files, it has no real–time requirements and pictures can be kept in an internal buffer as long as wanted. A preliminary idea was that frames are written to the output after they are no longer necessary for reference (inter–prediction). But this also does not hold in every case. For instance, in the Soccer sequence shown in figure 5.6, there is

a delay of two full GOPs before the first frame is written. This issue was not further investigated, because after this measurement a side–track was chosen: the improvement of choosing a GOP size adaptively. (chapter 6)

5.2.3 Used values

After analyzing the results, it was concluded that the maximum decoding times and the delay determined in the first measurement do not represent the actual computation time for decoding a frame. They are, more or less, a combination of processing time and the time a decoded frame stays in a buffer. For the model used in the demonstration application, it was concluded that with enough buffering, the processing times of a frame can be approximated with the average value. Instead of using the maximum values, the reference budget and the budget needed for decoding a specific quality level are based on the average decoding times shown in tables B.1 and B.2. The reference budget is an average of the budgets needed for decoding the reference streams (350). The default multiplication factor was set to 1.1. No measurements were done to determine this factor, but the value can be changed while running the program.

5.3 Demonstration program

Subsection 5.3.1 describes the architecture used to create the demonstration program. The observations done using the program are in subsection 5.3.2.

5.3.1 Architecture

The goal of the demonstration application is to show the use of scalable video. It is possible to show the difference between using scalable video or not and to make observations on the perceived quality. To represent a multipoint video conference, a mock–up conference is shown with six participants. It was chosen to use a layout where one screen is twice as big as the other screens (4CIF and CIF), as shown in 5.7. For each participant the same video stream is used to be able to compare the video of the speaker with the non–speakers. These streams are raw video files, stored on disc, decoded at the different quality levels.

When running the demo, it is possible to switch between using scalability or not and change the budget or multiplication factor. It also is possible to set the priority of the different quality levels to size→framerate→bitrate or framerate→size→bitrate. A full description of the design of demo application can be found in appendix C.

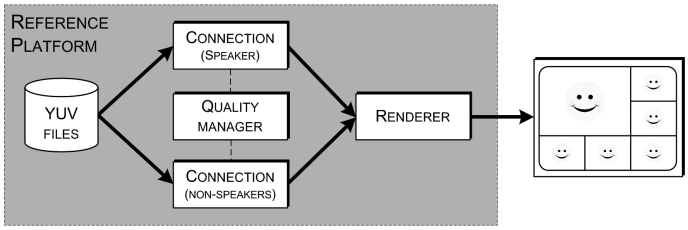


Figure 5.7: Conceptual dataflow demo application

The three most important blocks shown in figure 5.7 are highlighted here.

1. *Connection*
   This component represents a network layer capable of filtering. It reads raw video

files at different quality levels from disc. These files are previously encoded, extracted and decoded. There are two connection components in the demo program, one for the active speaker and one for the five non–speakers.

2. *Quality Manager*
   The quality manager instructs a connection component which quality level to read (extract). Each time a parameter changes, the quality manager determines an optimal quality level using the reference budget, multiplication factor and the budgets of the different quality levels. These budgets are based on the average decoding times determined in the first measurement.

3. *Renderer*
   The renderer combines the different video streams to one frame and outputs this to screen. Because the screen layout is fixed, it uses upscaling if necessary. This is necessary for scaling an input video to the desired size. Upscaling is done by duplicating pixels. The renderer also keeps track of timing, to make sure streams of different framerates are combined correctly.

5.3.2 Observations

The demonstration program shows that the use of scalable video is beneficial. When scalability is enabled, the perceived quality of the speaker increases, while there is no visible degradation in quality of the non-speakers[3]. The increase in quality of the speaker is less visible than expected. With the predefined budget the quality levels change from CIF@30–768 for every participant to 4CIF@30–2048 for the speaker and CIF@15–515 for the non–speakers. Thus, although a simple upscaling technique is used to scale the video for the speaker from CIF to 4CIF, this already is quite effective. Because spatial scalability has a large impact on the processing needed for decoding, it is advisable to use a (more sophisticated) upscaling technique if this requires less processing.

With respect to the difference in prioritizing the quality levels in size→framerate or framerate→size a second observation can be made. The videos with a higher framerate have a higher perceived quality than those with a lower framerate, but a higher picture quality. This confirms the observation mentioned in section 2.4, that smooth continuous video is more important. A video with a higher framerate looks smoother and has a higher perceived quality. This does require that the video is shown without jitter. It is recommended to use temporal scalability to highlight the active speaker.

Another observation of the demonstration program is that the use of a larger screen for the active speaker is effective in drawing the user focus. The effect of switching a user cannot be observed, because the same video was used for every participant. It is expected that switching will not degrade the perceived quality, as long as there is a feasible delay between two switches. It was also noted that a smaller screen, although shown on a low quality, can have a higher perceived picture quality than the larger screen. Thus, when using a larger screen for the active speaker, the quality of the non–speakers can even be lowered more, while the perceived quality does not degrade.

5.4 Conclusion

The feasibility of using scalable video on a resource constrained platform to show the active speaker best, has been shown in a demonstration program. In order to create

[3] Note that the perceived quality is based on personal observations. When developing a system, it is advisable to perform subjective user tests to measure the quality.

this program a model was created to represent the decoding of multiple streams on a resource constrained platform. To use this model, measurements were done on several videos encoded with the JSVM codec. The observations done on the measurements and demonstration program result in the following recommendation.

The difference in budget needed for the different spatial layers is quite large and the visible effect is limited. Spatial scalability can be used to facilitate different endpoints, but is not recommended to highlight the active speaker. The temporal and bitrate scalability can be used for highlighting the active speaker. This is further stated with the observation of the difference in prioritizing in size→framerate or framerate→size.

Note that the average decoding times used in the research model are a simplified representation of the processing needed to decode a frame. In reality, the decoding of different types of frames requires different processing times. The technique described in [15] uses bitrate scalability to adapt to these fluctuations. This can utilize the resources of a constrained platform even more efficient.

Chapter 6

Improving adaptive GOP

a·dapt (vb): "To make suitable to or fit for a specific use or situation."

This chapter describes an improvement of a specific part of the JSVM codec, the use of variable GOP sizes. Normally, the GOP size in the JSVM codec is fixed and is used to create the desired amount of temporal levels, as explained in chapter 3. Depending on the motion present in the sequence, it is possible to achieve better coding efficiency by using GOPs of variable sizes [17, 18]. The use of adaptive GOP can improve coding efficiency, for example in sequences where rigid motion or a scene change occurs.

The current algorithm used for creating variable GOP sizes during encoding can be improved by adding the pre–encoding of additional sub–GOPs. This improvement is explained by first describing the currently used algorithm in section 6.1. The motivation for altering this algorithm is explained in section 6.2. The proposed algorithm is discussed in section 6.3. Conclusions and possible future work with respect to this algorithm, are discussed in 6.4.

6.1 Current algorithm in JSVM

In the reference software of the JSVM encoder an input sequence is divided and processed in GOPs. The encoder uses GOP sizes that are a power of two, normally of size 4 to 64, but other values are possible. In JSVM the boundary of a GOP is an I–or P–frame, as shown in figure 6.1. Note that if the boundary of a GOP is an I–frame it is not predicted from previous frames and the corresponding arrow should be omitted.

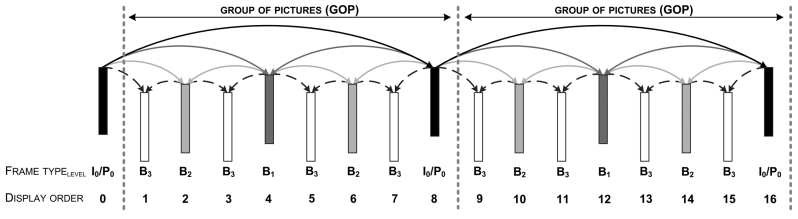


Figure 6.1: Hierarchical B–pictures and GOP size in JSVM

The creation of variable GOP sizes is done by pre–encoding a GOP in four different subGOPs shown in figure 6.2. From these subGOPs the PSNR is calculated and stored. These PSNR values are used in a mode–decision step to determine a division in subGOPs that results in the highest PSNR possible. This division found is used to encode

the full GOP. The re–encoding is needed because the sub-GOPs created during encoding are not stored and to create correct reference frames. The descibed steps result in the following algorithm [17]:

1. **Pre-encode**
   (a) Encode GOP of size N, calculate and store its PSNR
   (b) Encode N using two sub-GOPs of size N/2, calculate and store their PSNR
   (c) Encode N using four sub-GOPs of size N/4, calculate and store their PSNR
   (d) Encode N using eight sub-GOPs of size N/8, calculate and store their PSNR
2. **Mode-decision**
   (a) Compare the PSNR of a GOP with the average PSNR of the two equivalent sub-GOPs.
   (b) If the PSNR of the GOP is larger than the average PSNR of the underlying sub-GOPs store the GOP size and exit mode-decision for this GOP. Else apply mode-decision on the two underlying sub-GOPs, repeat until smallest sub-GOP is reached.
3. **Encode**, using determined GOP division.

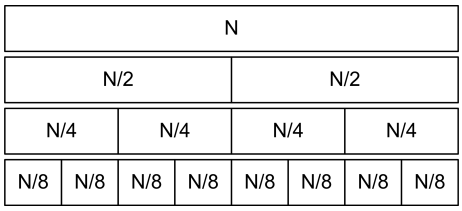


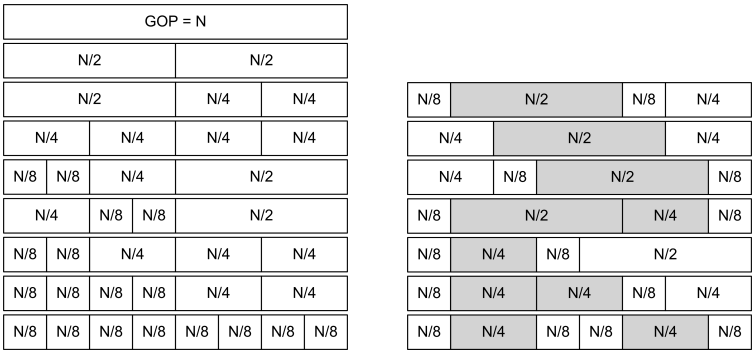
Figure 6.2: The different sub-GOPs used in JSVM for creating variable structure

6.2 Limitations of current algorithm

By using the PSNR values of the pre–encoded sub-GOPs as shown in figure 6.2, not every possible division of a GOP in sub-GOPs is taken into account. Figure 6.3(a) shows some (not all) possible examples of divisions in sub-GOPs that can be created using this algorithm. Examples of divisions (possible with the same sub-GOP sizes) that are not taken into account, are shown in figure 6.3(b). To be able to take these subdivisions into account, pre–encoding of the grey sub-GOPs is needed.

A second limit of the original algorithm is that it uses four, fixed decomposition steps to create the sub-GOPs. For small GOP sizes, such as 16, this results in a smallest sub-GOP of size 2. This limits the use of temporal scalability, because a frame in this small sub-GOP must be decoded for future reference. On the other hand, for large GOP sizes it might be useful to have more than four different GOP sizes.

To be able to create an *optimal division* in sub-GOPs and allow more flexibility in choosing the sub-GOP sizes the following alterations are proposed:



(a) Current (b) Not taken into account

Figure 6.3: Example of subdividing a GOP

- Include every possible sub-GOP within a GOP to determine an optimal division
  For example, if 4 different GOP sizes are used, the sub-GOPs shown in figure 6.4 are added to the sub-GOPs already shown in figure 6.2. The PSNR values of these sub-GOPs are also used in the mode–decision step.
- Use a flexible upper/lower bound for the GOP and sub-GOP sizes
  The maximum GOP size is determined by random access requirements. The lower bound is used for creating a minimum of temporal levels. These bounds allow choosing the amount of possible sub-GOP sizes more flexibly.

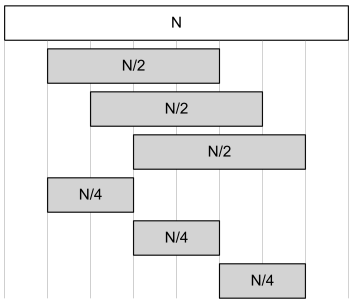


Figure 6.4: Example of added sub-GOPs

6.3 Proposed algorithm

The proposed algorithm creates an optimal division in possible sub-GOPs defined by the maximum and minimum GOP size. It still consists of the original three steps of pre–encoding, mode–decision and encoding. The pre–encoding and mode–decision steps are altered as followed:

1. **pre-encode**

   (a) `n = the first frame of the GOP`

   (b) `Encode every sub-GOP possible within n to N, starting with the largest possible, down to the smallest. Calculate and store the PSNR values for these sub-GOPs.`

```
    (c) If n == (N-smallest sub-GOP) proceed to mode-decision, else repeat
        step b with n = (n+smallest subGOP)

2. mode-decision

    (a) create a table with optimal PSNR for every possible subdivision
        and a second table with the boundaries where this division is
        optimal (explained in 6.3.2)

    (b) backtrack through the second table to find optimal division.

3. encode, using determined GOP division
```

This algorithm is capable of taking into account the extra subdivisions possible. This can improve the coding efficiency of the encoder. A more elaborate pseudo–algorithm of this proposal can be found in appendix D.

6.3.1 Pre–encoding

The loop shown ensures that every possible sub-GOP is encoded, keeping in mind the upper and lower bound. Secondly, the encoding is done in the order shown in figure 6.5. This ensures that when the encoding of a (sub)GOP starts, it can use the buffered reference frames previously stored. For each sub-GOP the PSNR is calculated and stored. Note that in the figure the maximum GOP size is N and the minimum is $N/8$, this can be altered.

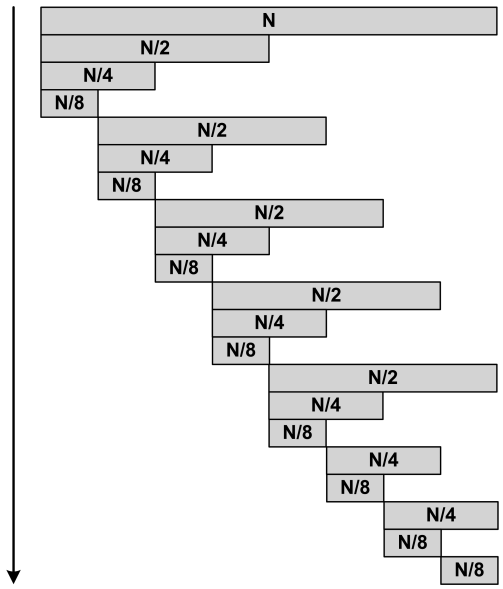


Figure 6.5: Graphical representations of the pre–encoding loop

6.3.2 Mode–decision

The mode–decision is divided in two steps, *filling tables* and *backtracking*. These steps, as explained next, are based on a technique called dynamic programming [6, cha.4]. Dynamic programming ensures that an optimal solution is found, within the given boundaries.

(a) Filling tables

First a table (or pyramid) with the highest PSNR possible for every subdivision possible has to be created. This is necessary, because when determining the division, subdivisions that cannot be encoded with one (sub)GOP also have to be considered. For example a sequence of six frames can be encoded using two sub-GOPs of size 2 and 4, but a GOP of size 6 does not exist.

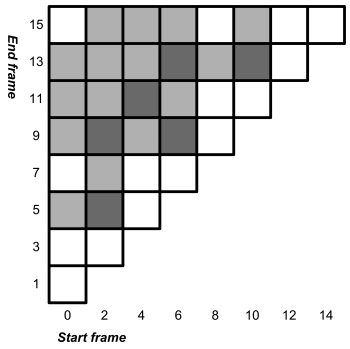


Figure 6.6: Table for storing optimal PSNR values

An example of a table used for storing PSNR values is shown in figure 6.6. Note that the framevalues are a multiple of the minimal desired GOP size. The white and dark grey areas are divisions that have to be pre–encoded in the previous step. All the grey areas represent possible divisions added, compared to the original algorithm. For each area in the table the highest PSNR value has to be calculated. This is done by starting at the bottom of the pyramid. For each area, the maximum PSNR is determined of every possible subdivision. While constructing this table with the optimal PSNR for each subdivision, a second table is used to store the frame where this subdivision is found. This value determines a frame boundary, where a GOP is 'split'.

(b) Backtracking

The second step consists of recursively backtracking through the table with frame boundaries to find the optimal sub-GOP division with the highest PSNR possible. This starts at the top of the pyramid and is applied for each subdivision found until the value of the boundary is the same as the end frame of a subdivision. The concept of this backtracking is shown in figure 6.7, where it results in the values [1, 9, 13, 15]. This resembles sub–GOPs of size [2, 8, 4, 2]

6.4 Conclusion and future work

An improved algorithm for creating variable GOP sizes adaptively is proposed and explained. Testing the results of using this algorithm is future work.[1] The first step is to determine the difference in quality/coding efficiency of the new algorithm. It is expected that the algorithm introduces an improvement. Secondly, it must be analyzed if the improvement is useful compared to the added encoding time. Because the use of variable GOP sizes is only useful for specific input videos, the added complexity might not be justifiable. If the testing results in a significant improvement, it is interesting to look at

[1] While testing the original algorithm it was found that the software first needed debugging and secondly that the algorithm can only be used for one spatial level. At that moment, including the new algorithm would take too much time.

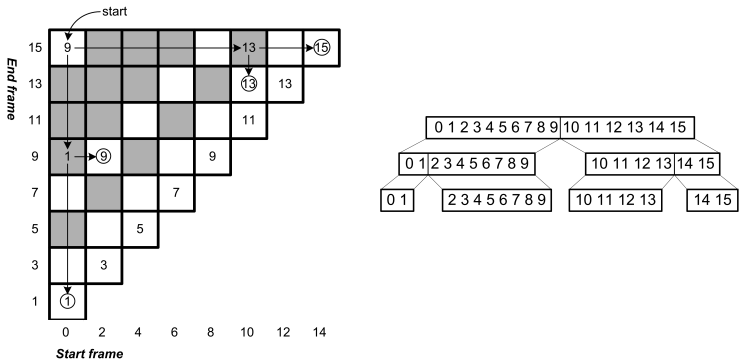


Figure 6.7: Example of backtracking through second table

other optimizations that are possible. For instance optimizing in speed.

It must also be noted that the proposed algorithm is an alteration of an existing method. Another approach is to look at a totally different method for creating variable GOP sizes. An example is by using statistical data of previously encoded subdivisions of several sequences. Such an approach can improve the encoding speed, as the search area is more limited.

Chapter 7

Conclusions and recommendations

> *con·clude (vb):* "To arrive at (a logical conclusion or end) by the process of reasoning; infer on the basis of convincing evidence"

Conclusions

To improve the quality of a multipoint video conference in the *fully connected world*, scalable video provides a solution. Together with several multicast channels efficient use of bandwidth is provided, while serving heterogenous terminals. It also provides means to trade resources, needed to for the non–speakers, to show the active speaker at a higher quality.

To limit the amount of multicast channels, filtering using payload identification of RTP can be used. Filtering is also possible in an environment, where multicasting is not available.

The scalable video encoder studied is the Joint Scalable Video Model (JSVM), an extension to AVC. Only a slow reference implementation of JSVM is currently available that cannot be used in an application with real–time requirements. The feasibility of using scalable video to optimize the use of available resources of a constrained platform is shown with a demonstration program. The program uses a created model of the available processing resources.

The demonstration program and measurements done on JSVM streams illustrate the benefit of using scalable video. They also show that temporal scalability introduces a structural delay in en- and decoding. In the demonstration program, the visible effect of using spatial scalability to highlight the active speaker is limited.

Next to the research done on multipoint video conferencing, an improvement of the JSVM encoder is proposed. The explained algorithm results in an improved picture quality/coding efficiency by creating an optimal division in the possible sub–GOPs. Precise results on the impact of this improvement were not obtained due to issues with the reference software.

Recommendations

With respect to a multipoint video conferencing application the use of scalable video can be recommended. The use of JSVM is preferable, as it is an extension to AVC, but an

optimized version is needed. The parameters of how many streams can be decoded depend on the resources of a target platform. For QoS control, a model is needed of the resources a specific quality level needs. Processing resources can be measured, networking resources depend on the maximum bit–rate. This bit–rate can be communicated during a session setup.

Spatial scalability should not be used to highlight the active speaker, as its effect is limited. But it can be used to allow terminals with different available resources to choose a (global) quality level. The use of multicasting is also recommended, but this is currently not widely available. For highlighting the active speaker, temporal and/or quality scalability can be used. Quality scalability can also be used to adapt to processing fluctuations. To determine parameters with respect to perceived quality it is advised to perform subjective user tests. This also includes testing the layout of the user interface. The used interface where one speaker is shown in a larger screen provided a good option for highlighting the active speaker.

The use of scalable video coding has several other potentials, not only for video conferencing. In a video broadcast setting, the choice in quality allows heterogenous terminals to view the video at the maximum quality possible using one stream. A terminal can also use scalable video for a picture–in–picture application, where two videos are shown simultaneously and the second video is shown smaller, within the first video. The smaller video can be an extraction of the original quality, as it is not shown at the full quality possible.

To test the improvement of the proposed algorithm it has to be included in the reference software. The results can be compared with the original algorithm and the use of a fixed GOP size. If the algorithm provides a significant improvement, it is interesting to look at improvements in speed. Another option is to look at an alternative way of creating variable GOP sizes.

Appendix A

Typical JSVM encoder

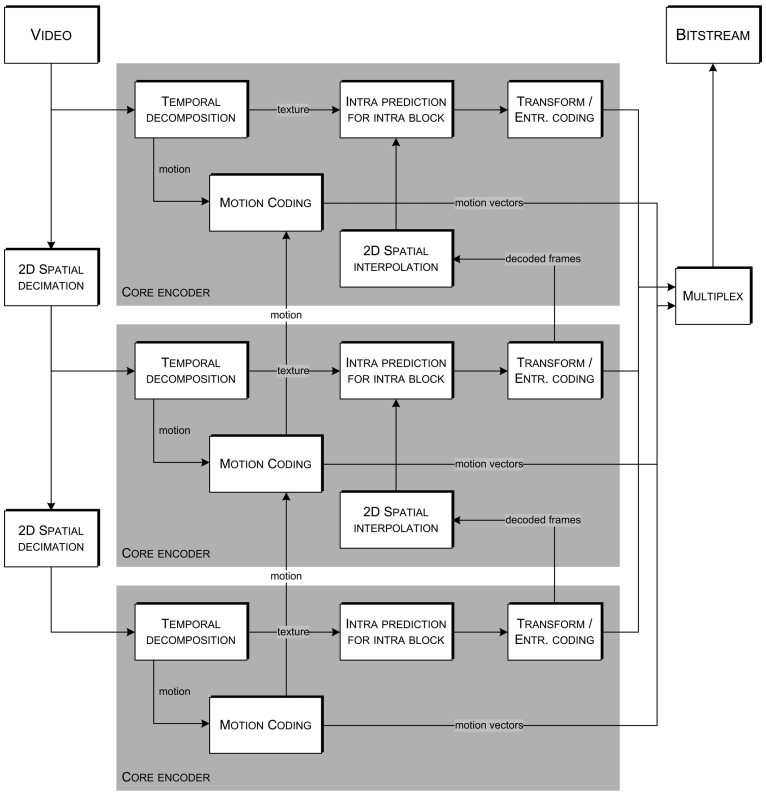


Figure A.1: Scalable JSVM encoder with 3 levels of spatial scalability

Figure A.1 shows an example of a scalable JSVM encoder. Spatial scalability is created by using three core encoders. The input to these encoders is the same sequence, but the 2D downsampling results in lower input resolutions for the encoders of a lower level. Within a core encoder motion estimation/compensation is applied, using hierarchical B–pictures. An encoder of a higher level uses the motion vectors and decoded pictures from a lower level to create a bitstream that is scalable with respect to this lower level. The motion vectors and residual data from the different levels are multiplexed to create a scalable bitstream as shown in chapter 3.

Appendix B

Measurement results

	Max. [ms]	Average [ms]	Delay [ms]	Max×fps
Crew	2.908	311	6.005	174.300
Harbour	17.619	419	30.898	1.057.140
Soccer	953	329	26.754	57.180

Table B.1: Decoding times reference videos (4CIF@60–3072)

	Max. [ms]	Average [ms]	Delay [ms]	Max×fps
QCIF@7.5–96	14	10	78	105
QCIF@15–192	28	16	127	420
CIF@7.5–192	64	55	409	480
CIF@7.5–288	76	60	445	570
CIF@7.5–384	84	73	535	630
CIF@15–256	130	57	481	1.950
CIF@15–384	143	65	560	2.145
CIF@15–512	155	68	612	2.325
CIF@30–384	143	59	515	4.290
CIF@30–576	165	62	562	4.950
CIF@30–768	197	98	875	5.910
4CIF@15–768	592	276	3.514	8.880
4CIF@15–1024	652	304	3.861	9.780
4CIF@15–1536	714	331	4.240	10.710
4CIF@30–1024	602	283	3.637	18.090
4CIF@30–1536	714	307	4.240	21.420
4CIF@30–2048	967	486	6.221	29.010

Table B.2: Decoding times conference videos

The measurements were done on a linux machine booted in a shell. (No X, few processes running in the background.) Each measurement is carried out eight times in order to average out measurement errors. The steps to create the conference streams are:

1. Capture video
2. Crop/cut video
   Needed to obtain 1200 frames of 4CIF@30fps.
3. Downscale
   The encoder does not scale the input video itself, this has to be done prior to encoding. This is needed to obtain CIF@30 and QCIF@15 input streams.

4. Encode (create one scalable bitstream)
   For encoding, the configuration files of the Soccer sequence were used and adjusted. The adjustments were the removal of reading motion info from a file, adjusting the GOP size and adjusting the intra period. The reference streams are encoded in several passes. These passes are used to obtain motion info and use this motion info for a next encoding pass. This method is not used for the video conference stream as this cannot be done in a real–time application. The video conference stream is encoded in one pass. The second adjustment was the change in GOP size. This size limits the amount of temporal layers available. It is set to four, to create three temporal layers. The final adjustment is in the intra–period. This is set to 32, in order to have an I–frame approximately every second at a framerate of 30fps. (and be able to switch to a higher level)

5. Extraction layers
   From the obtained bitstream, first the highest quality stream is extracted (4CIF@30fps–2048). From this stream the other levels are extracted. For this extraction a tool provided by the reference codec is used: BitStreamExtractor.

Algorithm 1 shows the pseudo code of the decoding process of a JSVM stream and added timer. By using integers t_1 and t_2, the time (in milliseconds) between the output of two consecutive frames is measured. With these measurements, the maximum, average decoding time and the delay are determined.

Algorithm 1 Pseudo code of decoding process (and added timers)

```
1:  t1 ← GetTick()
2:  while (!EOF) do
3:      extractPacket()
4:      processPacket()
5:      if (picture in buffer) then
6:          t2 ← GetTick() − t1
7:          append t2 to output file
8:          writeFrame()
9:          t1 ← GetTick()
10:     end if
11: end while
```

Appendix C

Design demo application

C.1 Introduction

This appendix describes the requirements of the aimed demo application, shows the architecture and the description of the components. For more in-depth information, doxygen documentation can be generated. The goal of the program is to interactively show the difference between the use of scalable video or not (in a multipoint videoconference). The program will show a mock up video conference screen as shown in figure C.1. During the program it will be possible to switch between the use of scalability or not, in order to show the difference.

To represent a target platform, a budget is used that can be spent on decoding several streams. This budget is based on the metrics described in chapter 5. Next to the budget a multiplication factor is used. The meaning of this factor x ($x > 1$) is the extra effort needed to decode more than one stream. *"The decoding of an additional stream requires x times the budget defined."*

C.2 User requirements

1. The application must allow switching between using scalability or not. [M]
   Most important, as this shows the difference.

2. The application must allow to upper/lower the budget of the reference platform. [M]

3. The application must allow to upper/lower the multiplication factor. [M]

4. It should be possible with the application to switch priority of determining the non-speaker level from size→framerate→bitrate to framerate→size→bitrate. [S]
   This can show the difference between using the highest possible spatial resolution but a low temporal versus lower spatial resolution but a higher framerate.

5. The application could allow to upper/lower quality level of speaker [C]
   (to free more budget for the non–speakers.)

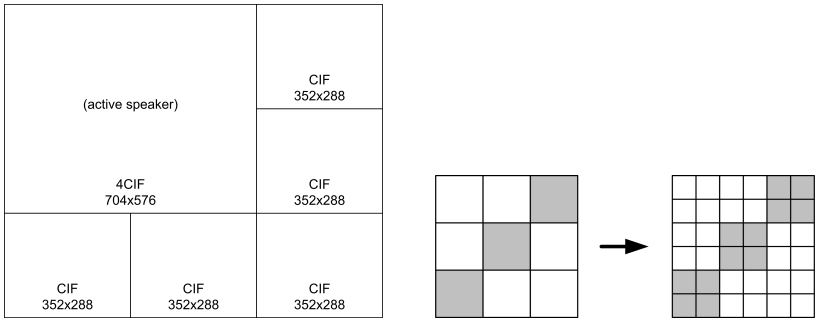
6. It could be possible to add/remove speakers. [C]

7. The application will not take your 'own' stream into account or show it. [W]
   It is assumed that the capturing and showing your own stream is outside the defined budget. Also the encoding of this stream is not taken into account in the budget.

(Prioritized using MoSCoW. (Must have, Should have, Could have, Won't have))

C.3 Functional requirements

1. The maximum output framerate of the video is 30fps
2. The used video layers are specified in table C.1.
3. An identical video stream will be used for each speaker
4. Files will be read from disk
5. The decoding times for different layers are known
6. The screen size is fixed to 1056×864 as shown in figure C.1(a). If a certain budget does not allow a spatial resolution needed, the input image will be upscaled by duplicating the pixels of an image as depicted in figure C.1(b). (Or even twice in the case of scaling from QCIF to 4CIF.)



(a) Screen layout (b) Upscaling

Figure C.1: Screen layout and upscaling

7. It will be possible to output the combined videoconference stream to disc. (If the combining and showing of several streams proves to be too slow, this output stream can be used to show the effect.)

C.4 Description

This section gives a short description of the demo program. For more in-depth information, a doxygen document can be generated from the source–code.
(`doxygen ../doc/MDVC.Doxyfile`).

C.4.1 User Input

s	Switch between using scalability or not.
up/down	Increase/decrease available budget.
left/right	Increase/decrease multiplication factor.
n/m	n = decrease max. quality speaker, m = increase
u/y	u = set priority to FPS→Spatial, y = Spatial→FPS
r	Reset filereader to frame 0
q/esc	Quit program
p	Pauze output
f	Advance one frame

Resolution	Framerate [fps]	Bitrate [kbit/s]
QCIF (176×144)	7.5	96
QCIF	15	192
CIF (352×288)	7.5	192
CIF	7.5	288
CIF	7.5	384
CIF	15	256
CIF	15	384
CIF	15	512
CIF	30	384
CIF	30	576
CIF	30	768
4CIF (704×576)	15	768
4CIF	15	1024
4CIF	15	1536
4CIF	30	1024
4CIF	30	1536
4CIF	30	2048

Table C.1: Conference video layers

C.4.2 Classes

The conceptual dataflow of the application is shown in figure C.2, the different classes are shown in figure C.3.

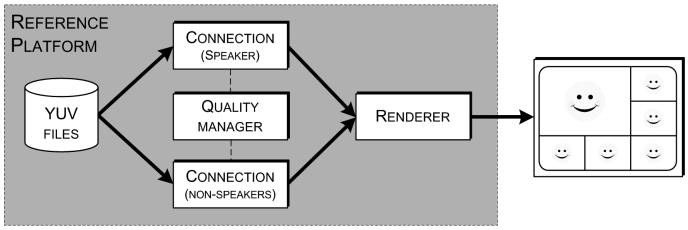


Figure C.2: Dataflow

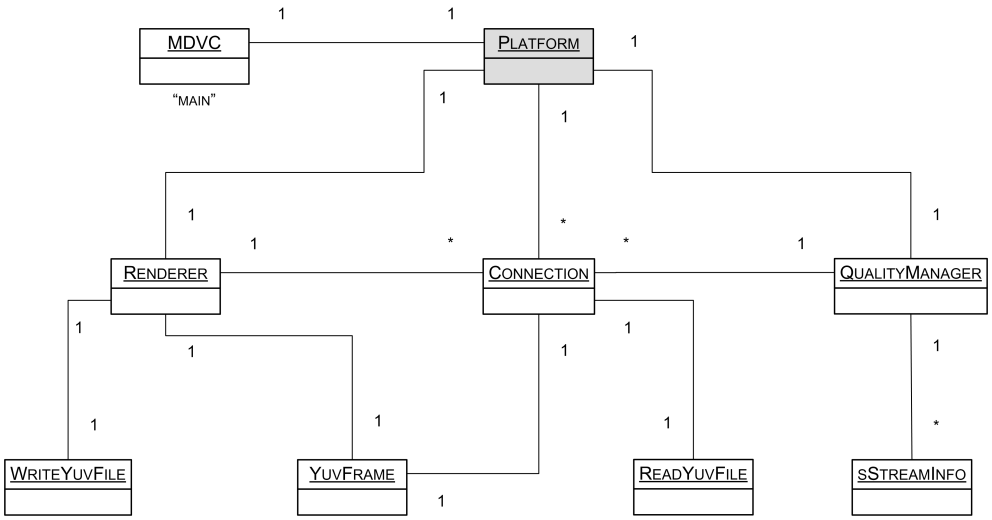


Figure C.3: Classes and connections

Platform	
Description:	Representation of a video conferencing platform. The platform is called from `MDVC.cpp`. It initializes two connections (one for the speaker and one for the non–speakers), the Renderer and the Quality manager. If the Platform is running, it is responsible for handling the user input.
public methods:	`init(), go()`
Connection	
Description:	Representation of a connected speaker. The most important functions of the connection class are grabbing a frame, set a quality level and set the stream to a desired frame number.
public methods:	`init(ScreenSize eSize, float fFrameRate, int iBitRate), grabFrame(), getScreenSize(), getFrameRate(), getBitRate(), setLayer(ScreenSize eSize, float fFrameRate, int iBitRate, int iFrameNr), setStream(int iFrameNr)`
Renderer	
Description:	The Renderer Class combines different video streams to one frame and outputs this to screen/file. Internal the Renderer uses upscaling if necessary. It also keeps track of timing, to make sure streams of different framerates can be combined. For displaying videos on screen, the Simple DirectMedia Layer (SDL) library is used [9].
public methods:	`init(Connection *pSpeaker, Connection *pNonSpeaker, bool bSDL), render(int iCurrentFrame), printInfo(int iBudget, float fFactor)`
QualityManager	
Description:	The quality manager searches for the optimal quality–level possible with a given budget/multiplication factor. This can be prioritized using screensize->framerate, or framerate->screensize. The budget, factor and use of scalability or not can be changed. The priority can be changed. The maximum quality of the speaker can be changed. The quality of the non–speakers is always less or equal to that of the speaker.
public methods:	`init(Connection *pcSpeaker, Connection *pcNonSpeaker, int iBudget, float fFactor), setBudget(int iBudget, int iCurrentFrame), setFactor(float fFactor, int iCurrentFrame), setScalable(bool bIsScalable, int iCurrentFrame), upSpeaker(int iCurrentFrame), downSpeaker(int iCurrentFrame), setPriorityFps(int iCurrentFrame), setPrioritySize(int iCurrentFrame)`
WriteYuvFile	
Description:	Can be used to write rendered frame to disk. Not implemented.
public methods:	–
ReadYuvFile	
Description:	Used to read different videoconferencing streams from disc. A file can be switched and a jump to a desired frame can be made. The filenames are in the format `Conference_[size]@[framerate]-[bitrate].yuv`
public methods:	`init(ScreenSize eSize, float fFrameRate, int iBitRate), readFrame(YuvFrame *pFrame), setYuvFile(ScreenSize eSize, float fFrameRate, int iBitRate, int iFrameNr), setFrame(ScreenSize eSize, float fFrameRate, int iFrameNr)`
YuvFrame	
Description:	Representation of an YUV frame in YV12. A class to store YUV frames. An YUV frame can be resized to desired width/height.
public methods:	`init(int width, int height), resize(int width, int height), getWidth(), getHeight()`
sStreamInfo	
Description:	Structure containing information of a video stream. The Quality manager uses an array of sStreamInfo to store the values of the available video streams.
public *attributes*:	ScreenSize eSize, float fFrameRate, int iBitRate, int iCost

C.4.3 Sequence Diagrams

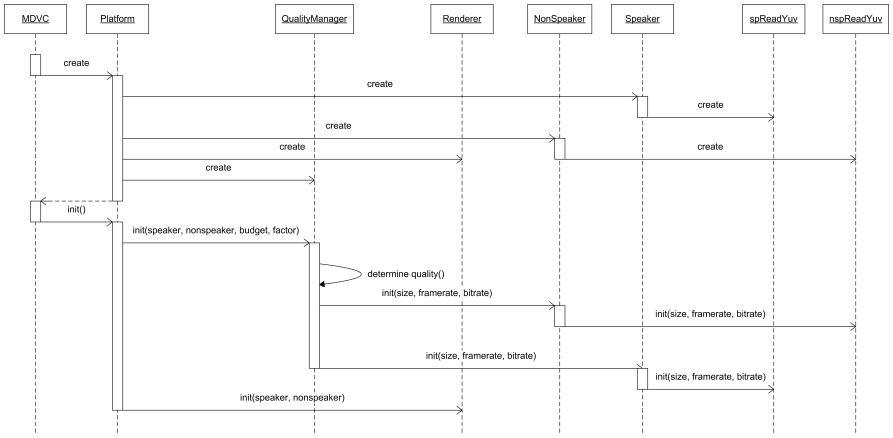


Figure C.4: The initialization of a platform

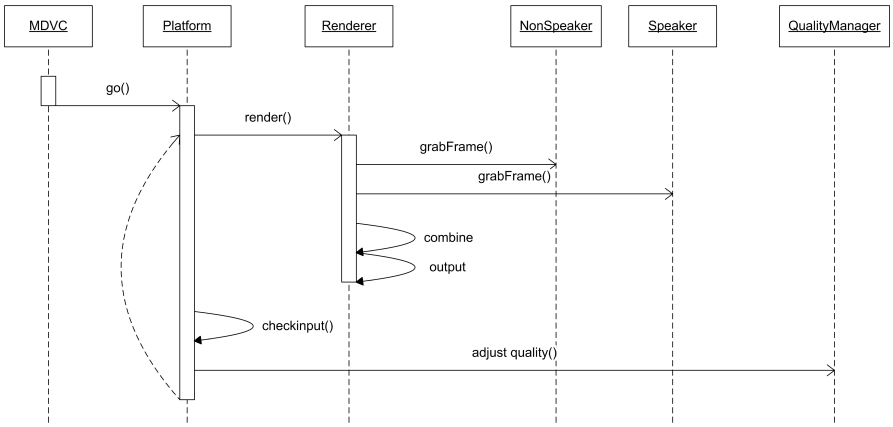


Figure C.5: Normal operation

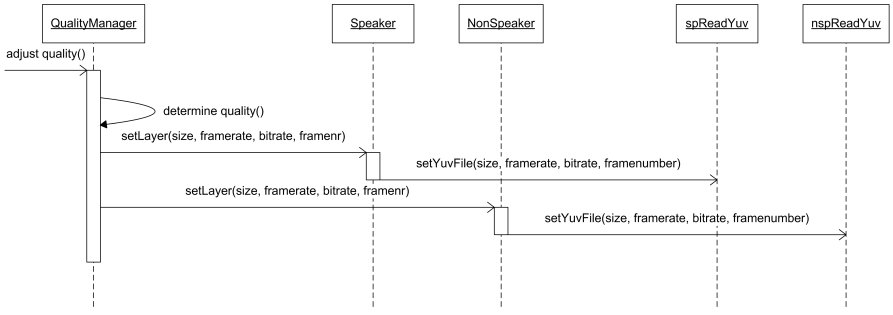


Figure C.6: Quality adjustment

The call 'adjustQuality' in C.6 refers to a change in available parameters. (e.g. budget, factor, etc.)

C.5 Known bugs/limitations

- The width of the output screen is cropped with 32 pixels at the right side of the screen. This is due to limitations of the SDL library. The maximum screen width possible is 1024 (not 1056)
- It is not possible to write combined frames to disc. Because the combining and output to screen was possible in real–time, WriteYuv is not implemented.
- When reaching the end of a file *and* a switch of file occurs, the largest files (4CIF @ 30fps) are not displayed correctly. Suspected errors are a rounding fault in setting file read pointer, or a problem with the maximum size of an integer in setting file pointer.
- The program assumes six connections (1 speaker + 5 non–speakers). In some instances the use of these values are hard coded in the source, thus not portable.
- Due to above reason, it is currently not possible to add/remove connections.

C.6 Conclusion

The demo program shows the conceptual use of SVC in a multipoint video conference. For a real application it is advised to determine a maximum and minimum of the required frame size, frame rate and bitrate. Within this maximum/minimum a layering can be determined. For frame size and frame rate, extraction points have to be defined, bitrate can be truncated to an arbitrary level.

Appendix D

Pseudo code algorithm

This appendix shows the pseudo code of the proposed algorithm as explained in chapter 6. For the assignment of a value to a variable, $\leftarrow$ is used. The comparative operators used are $<, >, =$. Note that the variables used in the different for–loops are not in- or decremented by 1, but with the minimal GOP size.

D.1 Encode GOP

This is the main algorithm for applying adaptive GOP on a GOP of size GS_{max}. The minimal used subGOP size is GS_{min}. The algorithm consists of three main steps, pre–encoding, mode–decision and the final encoding.

Algorithm 2 $Encode$GOP(GOP, GS_{min}, GS_{max})

```
1: create table for storing PSNR and fill with 0s
2: PreEncode(GOP, PSNR, GS_min, GS_max)
3: division[] ← ModeDecision(PSNR)
4: Encode(division)
```

D.2 pre–encode

This algorithm encodes every different subGOP possible within a GOP of size GS_{max} and with a minimal GOP size of GS_{min}.

Algorithm 3 $PreEncode$(GOP, PSNR, GS_{min}, GS_{max})

```
1: for i ← 0 to i = GS_max − GSmin do
2:     j ← i + GS_max
3:     while j ≥ GS_max do
4:         j ← j − GS_min
5:     end while
6:     while j > i do
7:         encode GOP i to j and calculate/store PSNR[i, j]
8:         j ← j − GS_min
9:     end while
10:    i ← i + GS_min
11: end for
```

D.3 mode–decision

The mode–decision algorithm determines the optimal GOP division, by first calculating the highest PSNR possible for each subdivision and secondly use backtracking to determine the boundaries, where a GOP is 'split'.

Algorithm 4 $ModeDecision(\texttt{PSNR})$

```
1: create table F for storing boundaries
2: {Fill the tables}
3: for (j ← 1 to j = GS_max) do
4:    for (i ← j − 1 downto 0) do
5:       minPSNR ← PSNR[i, j]
6:       F[i, j] ← k
7:       for (k ← i to j) do
8:          if ((PSNR[i, k] + PSNR[k, j])/2 > minPSNR) then
9:             minPSNR ← (PSNR[i, k] + PSNR[k, j])/2
10:            PSNR[i, j] ← minPSNR
11:            F[i, j] ← k
12:         end if{increase k with GS_min}
13:      end for{decrease i with GS_min}
14:   end for{increase j with GS_min}
15: end for
16: return Backtrack(F, 0, GS_max)
```

Algorithm 5 $Backtrack(\texttt{F}, i, j)$

```
1: if F[i, j] = j then
2:    return F[i, j]
3: else
4:    Backtrack(F, i, F[i, j])
5:    Backtrack(F, F[i, j] + 1, j)
6: end if
```

Acronyms and abbreviations

3GPP 3rd Generation Partnership Project

4CIF Four–CIF (704 × 576)

AVC Advanced Video Codec

CIF Common Interchange Format 352 × 288

DVB Digital Video Broadcast

fps frames per second

GOP Group of Pictures

HHI Heinrich–Hertz–Institut

IEC International Electrotechnical Commission

IETF Internet Engineering Task Force

IP Internet Protocol

ISO International Organization for Standardization

ITU International Telecommunication Union

JSVM Joint Scalable Video Model

JVT Joint Video Team

LAN Local Area Network

MCTF Motion Compensated Temporal Filtering

MPEG Motion Pictures Expert Group

MSE mean-square-error

NAL Network Abstraction Layer

NALU NAL unit

PC personal computer

PSNR peak signal-to-noise ratio

QCIF Quarter–CIF (176 × 144)

QoS Quality of Service

RTCP Real-Time Control Protocol

RTP Real-time Transport Protocol

SDL Simple DirectMedia Layer

SIP Session Initiation Protocol

SSIM Structural SIMilarity

SVC Scalable Video Coding

TCP Transmission Control Protocol

TSE Technical Software Engineering

UDP User Datagram Protocol

VCEG Video Coding Experts Group

VCL Video Coding Layer

VoIP Voice over IP

VQM Video Quality Metric

Glossary

B–frame An inter coded frame using forward and/or backward predictions.

codec A combination of compressor and decompressor for audio/video.

current frame The current frame to be encoded by an encoder.

display order The original order of the different frames from a video sequence; The order in which video frames are shown.

doxygen A documentation system for C++, C, Java, Objective-C, Python, IDL (Corba and Microsoft flavors) and to some extent PHP, C#, and D. [11].

encoding order The order in which different video frames of a sequence are encoded (and transmitted).

entropy coding Compressing digital data exploiting frequently occurring patterns in bitstreams.

frame A rectangle of $n \times m$ pixels.

I–frame An intra coded frame.

inter frame coding Compressing a video sequences exploiting redundancy between adjacent frames.

intra frame coding Compressing a video frame exploiting redundancy between pixels.

macroblock A sample region of $x \times y$ pixels, most often 16×16.

MoSCoW Prioritizing requirements using: Must have, Should have, Could have, Won't have.

P–frame An inter code frame using only forward prediction.

pixel A picture element, one dot/point in a frame.

protocol A special set of rules that endpoints in a telecommunication connection use, when they communicate.

reference frame A previously encoded and subsequent decoded frame. Reference frames are used encoding a new frame usint inter prediction.

residual frame The result of substracting the prediction of a frame from this frame.

spatial resolution The amount of pixels in a frame, noted as $x \times y$ pixels.

temporal resolution The amount of frames per time interval, often noted in frames per second (FPS) or Hz.

video A sequence of frames.

References

[1] Image processing department HHI. http://ip.hhi.de/imagecom_G1/savce/.

[2] B. Barmada, M.M. Ghandi, E.V. Jones, and M. Ghanbari. Prioritized transmission of data partitioned H.264 video with hierarchical QAM. *Signal Processing Letters, IEEE*, 12(8):577–580, Aug. 2005.

[3] L. Bass, P. Clements, and R. Kazman. *Software Architecture in Practice, Second Edition*. Addison-Wesley Professional, April 2003.

[4] I. Beier and H. Koenig. GCSVA - A multiparty videoconferencing system with distributed group and QoS management. In *IC3N '98: Proceedings of the International Conference on Computer Communications and Networks*, page 594. IEEE Computer Society, 1998.

[5] P. Calyam, M. Sridharan, W. Mandrawa, and P. Schopis. Performance measurement and analysis of H.323 traffic. In *PAM*, volume 3015 of *Lecture Notes in Computer Science*, pages 137–146. Springer, 2004.

[6] T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein. *Introduction to Algorithms, Second Edition*. The MIT Press, September 2001.

[7] S.E. Deering. *RFC1112 - Host extensions for IP multicasting*. RFC Editor, http://www.rfc-editor.org/, 1989.

[8] D. Estrin, D. Farinacci, A. Helmy, D. Thaler, S. Deering, M. Handley, V. Jacobson, C. Liu, P. Sharma, and L. Wei. *RFC2117 - Protocol Independent Multicast-Sparse Mode (PIM-SM): Protocol Specification*. RFC Editor, http://www.rfc-editor.org/, 1997.

[9] S. Lantinga et al. Simple DirectMedia Layer. http://www.libsdl.org/.

[10] F. Halsall. *Data communications, computer networks and open systems (4th ed.)*. Addison Wesley Longman Publishing Co., Inc., 1995.

[11] D. van Heesch. Doxygen documentation system. http://www.doxygen.org/.

[12] C. Hentschel, R.J. Bril, Y. Chen, R. Braspenning, and T.-H. Lan. Video quality-of-service for consumer terminals - a novel system for programmable components. *International Conference on Consumer Electronics, Digest of Technical Papers*, pages 28–29, June 2002.

[13] Packetizer Inc. A primer on the H.323 series standard. http://www.packetizer.com/voip/h323/papers/primer.

[14] ITU Telecom. Standardization Sector of ITU. *Methodology for the Subjective Assessment of the Quality of Television Pictures, Recommendation ITU-R BT.500-11*, 2002.

[15] D. Jarnikov, P. van der Stok, and C.C. Wüst. Predictive control of video quality under fluctuating bandwidth conditions. In *ICME*, pages 1051–1054. IEEE, 2004.

[16] LogicaCMG. uOne™ Media Services Solution Platform. http://www.logicacmg.com/Telecoms/100020.

[17] L.P. Natário and F. Pereira. Adaptive GOP Size in the MPEG-4 Scalable Video Coding. In *Conf. on Speech and Image Processing, Multimedia Communications and Services - EURASIP*, 2005.

[18] G.H. Park, M.W. Park, S. Jeong, K. Kim, and J. Hong. Improve SVC Coding Efficiency by Adaptive GOP structure. (jvt-O018.doc). http://ftp3.itu.ch/av-arch/jvt-site/2005_04_Busan/JVT-O018.doc.

[19] C. Perkins. *RTP: Audio and Video for the Internet*. Addison-Wesley, Jun 2003.

[20] B. Plattner, G. Parissidis, and K. Katrinis. A comparison of frameworks for multimedia, Jan 2004.

[21] J. Reichel, H. Schwarz, and M. Wien. Draft of Scalable Video Coding – Working Draft 4. (jvt-Q201). http://ftp3.itu.ch/av-arch/jvt-site/2005_10_Nice/JVT-Q201d1.zip.

[22] J. Reichel, H. Schwarz, and M. Wien. Joint Scalable Video Model JSVM–4. (jvt-Q202). http://ftp3.itu.ch/av-arch/jvt-site/2005_10_Nice/JVT-Q202.zip.

[23] I.E.G. Richardson. *H.264 and MPEG-4 Video compression, Video coding for the next generation*. Wiley, 2003.

[24] H. Schulzrinne and J. Rosenberg. A comparison of SIP and H.323 for internet telephony. In *Proceedings of Int. Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV)*, pages 83–86, July 1998.

[25] D. Sisalem and J. Kuthan. SIP tutorial, Apr 2001. http://www.iptel.org/sip/.

[26] T. Stockhammer, M.M. Hannuksela, and T. Wiegand. H.264/AVC in wireless environments. *IEEE Trans. Circuits Syst. Video Techn.*, 13(7):657–673, 2003.

[27] A.S. Tanenbaum. *Computernetwerken (4de ed.)*. Prentice-Hall, Inc., 2003.

[28] Joint Video Team. Development archive. http://ftp3.itu.ch/av-arch/jvt-site/.

[29] J. Toga and H. ElGebaly. Demystifying multimedia conferencing over the Internet using the H.323 set of standards. *Intel Technology Journal*, 2:11, 1998.

[30] Columbia University. General SIP information. http://www.cs.columbia.edu/sip/.

[31] T. Wiegand, G.J. Sullivan, G. Bjntegaard, and A. Luthra. Overview of the H.264/AVC video coding standard. *IEEE Trans. Circuits Syst. Video Techn.*, 13(7):560–576, 2003.