Multidimensional Transcoding for Adaptive Video Streaming

Jens Brandt Lars Wolf

IBR
Technische Universität Braunschweig
Mühlenpfordtstrasse 23, 38106 Braunschweig, Germany
{brandt|wolf}@ibr.cs.tu-bs.de

ABSTRACT

Video content providers often offer their video streams at three or more different quality levels: low, medium and high quality. These different levels are needed to support different target devices. However, this static kind of adaptation to discrete quality levels cannot meet the requirements of all existing multimedia devices. Because of the increasing heterogeneity of Internet-enabled video devices a more dynamic and individual adaptation to the requirements of the consuming device is needed. One approach for video adaptation is the use of video transcoding techniques, where modifications of the video stream are done in the compressed domain to save processing power. Many transcoding approaches for digital video can be found in the literature, but these focus on adaptation in one single dimension only, such as temporal adaptation, spatial adaptation or quality adaptation. To support as many video-enabled devices with different capabilities as possible, adaptation in more than one dimension is needed. In this paper we present a multidimensional transcoding approach for MPEG-4 encoded video, which smartly combines existing transcoding techniques to enable fine grain adaptation for different video devices.

1. INTRODUCTION

The prominence of digital video on the Internet is rising constantly. But transmission as well as decoding of digital video streams still has high resource requirements. To meet the requirements of different target devices, video content providers have to offer their video streams at different quality levels. Due to the increasing heterogeneity of Internet-enabled video devices a static adaptation cannot meet the requirements of all devices. Especially when looking at mobile devices, a fine grain adaptation to the capabilities of the device is mandatory. The possibilities of video playback with mobile devices are mainly restricted by the limitation of the following resources: network bandwidth, processing power, display resolution, memory size and remaining energy. Although mobile video devices are getting more and more powerful, the processor speed of current PDAs, smart phones or pocket multimedia players is only about 300 – 600 MHz and their memory size is about 64 MB. Without special video decoding hardware such devices are often incapable of decoding and displaying high quality video streams from the Internet. This problem will remain in the near future, also since quality demands are rising as well.

NOSSDAV '07 Urbana, Illinois USA

	Bit Rate	Processor	Memory	Energy
Spatial Resolution	x	x	x	x
Temporal Resolution	x	x		x
Detail Resolution	x	x		x

Table 1: Impact of encoding parameters

Table 1 shows the impact of different video coding parameters i. e., spatial, temporal and detail resolution, on the resource requirements which the decoding device has to satisfy. In this table as well as in the rest of this work we focus on block-based video codecs that use motion compensation (MC) and discrete cosine transfer (DCT) for video compression. Examples of such video coding schemes are MPEG-1/2/4, H.261, H.263 and H.264. All three parameters that are given in the table directly affect the number of coded bits in the stream and therefore also affect the resource requirements on the decoding device. Especially the spatial resolution has an impact on all four resources. When reducing the spatial resolution of a video stream, its bit rate is reduced, which in turn also reduces the necessary network bandwidth and processing power to decode the video. Furthermore, because of smaller frame resolutions, the amount of memory needed to buffer reference frames as well as the consumed energy of the device are also reduced. Mobile devices often have a small screen resolution, thus the spatial resolution needs to be tailored to the screen resolution anyway. Further bit rate reduction can be achieved by reducing the temporal resolution (frame rate) and decreasing the detail resolution (increasing the quantization factor). Because of the reduced number of frames and the reduced number of coded DCT values, both reductions also have an impact on the processing power needed for decoding. Additionally, all parameters have an impact on energy consumption. In [10] it was shown that the energy consumption of video decoding can be reduced to 75 – 58% by increasing the quantization level which results in a quality decrease of just 5 – 13%.

In the literature mainly two different approaches for video adaptation can be found: scalable video coding and video transcoding. With scalable video coding the video stream is separated into a base and one or more enhancement layers. The base layer contains the complete stream in a low quality and each additionally received layer increases the quality of the video. If an adaptation is needed, those layers exceeding the capabilities of the device can be discarded, either on the device itself or in the network. In our work we concentrate on single layer video streams, because of the lower complexity needed for their encoding and decoding compared to multi layer video streams. Furthermore, most of the available video streams on the Internet are single layer videos.

Video transcoding is a well-known mechanism for the adaptation of video coding parameters which are defined at encoding time. Compared to video recoding, where the video is completely decoded before it is encoded again, video transcoding works in the compressed domain. Thus, processing time can be saved at the expense of lower flexibility. Several specialized transcoding mechanisms exist, but hardly any of them focuses on transcoding in more than one dimension. In this paper we present a novel architecture for multidimensional transcoding, which we have implemented for MPEG-4 video streams. This architecture is intended to be used in our multimedia gateway system to support adaptive video streaming for mobile clients, which we have already presented in previous work [2].

The remainder of this paper is organized as follows: In section 2 we give an overview of existing transcoding techniques and related work. Afterwards we present our processing architecture for multidimensional transcoding in section 3. In section 4 we describe our multidimensional transcoding approach and implementation for MPEG-4 video, based on the aforementioned architecture. The evaluation of our implementation is presented in section 5. Finally we conclude this paper in section 6.

2. VIDEO TRANSCODING

The field of video transcoding has been explored in several recent studies. Most of them focus on different specialized transcoding techniques. A comprehensive overview of the research area dealing with those transcoding techniques is given in [13] and [7]. However, almost all of the presented approaches in the literature focus on adaptation of one single parameter, such as the three aforementioned, the bit stream syntax or the semantics of the content. Those coding parameters can also be interpreted as different dimensions in which the input video can be adapted. To handle the huge variation of resource constraints and users' preferences a multidimensional adaptation is needed. Adaptation along the spatial dimension is used to tailor the resolution of a video stream to the display resolution of the requesting device. This also reduces the memory and processor requirements on the device. Adaptation in the temporal dimension as well as adaptation in the dimension of detail resolution can be used for further bandwidth reduction.

Due to the compressed nature of digital video, the pixels of the video frames cannot be manipulated directly. Additionally, the reduction of the frame rate is not trivial, because of the dependencies between consecutive frames, caused by motion compensation. In digital video only some frames are encoded as complete frames i. e., key frames. The majority of the encoded frames use previous and future frames as reference frames, so that only the differences to those frames need to be encoded (i. e. P- and B-frames). The most obvious way to adapt compressed videos is the use of the cascaded pixel domain transcoder (CPDT), which simply consists of a cascaded decoder and encoder. In fact, the CPDT is a video recoder as mentioned before. But a complete decoding and encoding is quite inefficient. Moreover, the information that is already available from the previous encoding should be reused for transcoding to enhance the produced quality of the video. Most video coding standards such as MPEG video use the discrete cosine transfer (DCT) to transform each frame from the pixel domain into the frequency domain. In a CPDT this transformation needs to be performed from the frequency domain and back, which is computationally very intensive. The main idea of video transcoding is the avoidance of this complete decoding and encoding. Mathematically, the DCT is a linear and orthogonal transformation which is distributive with respect to matrix multiplications. To avoid the computation of the DCT we can take advantage of these characteristics of the DCT [3]: To manipulate a video block A we need two matrices X and Y to compute the manipulated block $B = X \cdot A \cdot Y$. Due to the linearity of the DCT we have:

$$DCT(B) = DCT(X \cdot A \cdot Y) = DCT(X) \cdot DCT(A) \cdot DCT(Y)$$

Thus we can realize every linear pixel manipulation in the frequency domain. However, in most digital video formats motion compensation is used to utilize temporal dependencies between frames for compression. These existing dependencies are encoded by motion vectors which are pointing to a block in a reference frame and by the resulting error block, i. e. the difference between the current and the referenced block. Thus, to be able to manipulate a coded block in the frequency domain we have to compute the inverse motion compensation (IMC) in the frequency domain, which was firstly presented in [3] and subsequently optimized in further publications, for instance in [12] and [9]. These mechanisms were used in several proposed transcoding architectures, such as a frequency domain requantization transcoder proposed in [1], a frame-skipping transcoder proposed in [5] or in a spatial resolution downconversion transcoder in [8] and [6]. The only approach we are aware of that addresses transcoding in more than one parameter was proposed by Shanableh et. al. in [11]. They focused on heterogeneous transcoding of MPEG-1 video into H.261 / H.263 video with spatial and temporal resolution reduction. However, the reduction of each single parameter was treated separately and no architecture for the combined transcoding of the examined parameters was presented. In summary, an integrated approach for multidimensional video transcoding, especially for mobile devices has not yet been proposed.

In this work we concentrate on transcoding in three different dimensions which we have identified to be the most important for video adaptation for mobile devices: spatial resolution, temporal resolution and detail resolution. By transcoding in these three dimensions we can support fine grain video adaptation for mobile devices. However, our approach is not limited to these dimensions but can easily be extended to support adaptation also in other dimensions, such as heterogeneous transcoding from one video format to another.

3. PROCESSING ARCHITECTURE

Starting from the studies of different transcoding mechanisms for video adaptation in one single dimension, we have developed a multidimensional transcoding architecture. This architecture is based on the idea of smartly combining one-dimensional transcoders into a transcoder chain, which in turn forms a multidimensional transcoder. As illustrated in figure 1, in this architecture a transcoder T_i is a module which can consume and produce video frames. Such a transcoder consumes a frame only if another transcoder or program module requests a frame from it. In other words, each transcoder chain needs exactly one active component which controls the whole transcoding process. In the figure this is done by the controller, which is also responsible for the setup and configuration of the whole transcoder chain. Therefore, each transcoder in the chain can generate events e_i to notify the controller about its current state. In order to transcode a video stream by the shown chain, the controller requests a frame from the last transcoder in the chain (T_2) which consumes the frame from T_1 which reads the original stream from one or more video sources. Examples of such sources are files, network streams or streams from a live video source. The frames which the controller requests from T_2 are written to an output stream. The transcoder T_2 in the figure is a combination of two other transcoders T_{21} and T_{22}.

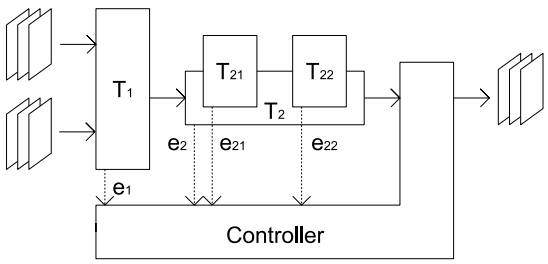

Figure 1: Transcoder Architecture - Overview

The video frames passed from one transcoder to another are typically not completely decoded, but they may be partially decoded. To indicate the actual coding state of a frame we have defined the following five states: *encoded, partially decoded, quantized, dequantized* and *IMC*. These states will be described in detail in the following section. If two or more transcoders are connected to setup a transcoder chain, the states of the frames produced by each transcoder have to match to the frame states, which its successor in the transcoder chain accepts.

This modular processing architecture provides good flexibility as well as good control of the whole transcoding process. During this process, each transcoder can generate events (e_i in the figure) to notify other program modules, e. g. the controller. Program modules can listen to these events and can firstly react to and secondly control each step of the transcoding process. These events are especially needed to inform the controller about the status of each frame, so that it can fully control each transcoding step of the frame.

4. MULTIDIMENSIONAL TRANSCODING

Since the focus of our research is dynamic video adaptation for mobile devices, we developed a multidimensional MPEG-4 transcoder for mobile devices, which uses the processing architecture presented in the previous section. Mobile devices often have a small screen resolution, which makes the use of spatial adaptation mandatory. Fortunately, this also reduces the bit rate and therefore the required bandwidth of the video stream. In order to achieve further reduction of the bandwidth of the video we decided to tailor the frame rate by the use of temporal adaptation and finally adjust the bandwidth of the stream by changing the quantization factor. The order in which the video should be transcoded in the three dimensions is quite obvious: Firstly those frames which are not needed in the resulting video should be dropped. Secondly the resulting frames should be scaled to the designated resolution and finally the DCT values of the remaining frames should be quantized again.

A simple combination of the existing transcoding mechanisms is not efficient. Some steps, such as the inverse motion compensation (IMC), are needed in all transcoders and therefore should be computed only once. While some transcoding mechanisms in the literature may not need the IMC, we have identified that in the case of multidimensional transcoding the IMC is always needed. For instance, when dropping P-frames new motion vectors have to be computed. Additionally, the error blocks coded in the remaining frames have to be recalculated by the use of this new motion vectors. Therefore, when working in the compressed domain, the DCT values of macro blocks of different frames has to be added. However, in MPEG-4 each macro block may have been encoded with its own quantization factor, which makes a direct addition of different macro blocks impossible. Moreover, when changing the DCT values of a frame which is used as a reference frame, the decoding of the referencing frame becomes inaccurate, because the encoded error blocks in the referencing frame were calculated based on the original DCT values of the reference frame. This inaccuracy is called drift error and can be avoided by using a drift free loop, which performs the IMC for each inter-coded block and eliminates the occurring error from the current block.

Only in the case of B-frame dropping the IMC can be avoided, because B-frames are never used as reference frames and therefore can be dropped without any influence on other frames. Thus, to drop a B-frame we only need to decode the type of the frame. Afterwards we can compute the IMC for the remaining frames which removes any temporal dependencies between the frames. However, the motion vectors are still saved in each frame for future use. Now, we can go on with dropping P-frames.

When dropping P-frames, motion information gets lost and motion vectors of subsequent P- or B-frames become invalid. Thus, we need to define new motion vectors for the following frames. Instead of searching new motion vectors, we can use one of the mechanisms for redefining motion vectors described in the literature, which were already mentioned in section 2. As the number of frames has already been reduced we can now reduce the spatial resolution of the remaining frames by using an existing spatial reduction method from the literature, e. g. the one proposed in [4]. When downscaling a video frame all motion vectors of this frame (in case of a P- or B-frame) as well as other motion vectors which are referencing to this frame (in case of an I- or P-frame) get invalid, because they are still pointing to the original region in the reference frame. Thus, also the coefficients of the motion vectors have to be downscaled. Ad-

ditionally, we have to compute one new macro block from several existing macro blocks, because macro blocks have a fixed resolution (e. g. 16×16 Pixel in MPEG video). But also the number of motion vectors of a macro block is fixed and we may have to create one downscaled motion vector by interpolation of several existing vectors. In the case of MPEG-4 we can benefit from the ability of MPEG-4 to use four vectors per macro block. For instance, if a video is downscaled by a factor of two and if each macro block has exactly one motion vector, we do not need to interpolate them but can use all of them in the new macro block. Afterwards we have to compute the motion compensation (MC) by using the newly created motion vectors. Within the MC we can either use the same quantization factor as before or a new one for further compression.

4.1 Frame Decoding

For communication between the transcoders in the chain, we have created a data structure which contains all information about the frame as well as its data. The encoding state of the frame indicates how much of the frame has already been decoded. Frames in the state *encoded* are completely encoded frames, as they are read from the input stream. After parsing the first few bytes of a frame, we get some information such as the type of this frame, and its state changes to *partially decoded*. When we further decode the frame, we get the quantized values of the DCT coefficients and the state changes to *quantized*. The next step of decoding is to dequantize the DCT values, which changes the state of the frame to *dequantized*. The final decoding step needed for transcoding is the computation of the IMC, after which the state of the frame is changed to *IMC*. Figure 2 shows the state chart of the frame states.

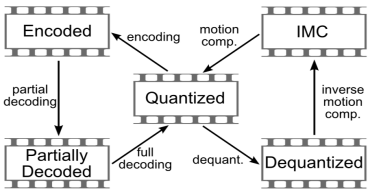

Figure 2: Frame States

4.2 Transcoder Chain

The complete transcoder chain is shown in figure 3. Not only are the modules which effectively transcode the video frames implemented as a transcoder but also those modules which are needed to pre- and post-process the video frames. At the beginning of the chain the PartialDecoder reads the video stream from the input and creates the data structure for each frame. Only the information needed to identify the frame type is decoded and saved in the data structure. Because dropping of B-frames does not affect any other frames, the BFrameDropper is placed right behind the PartialDecoder. It can read the type of a frame and drop those frames which are identified as B-frames. Afterwards the StreamDecoder is used to decode the quantized DCT values as well as the motion vectors (in the case of intra-coded frames) of each frame. The Dequantizer computes the inverse quantization of the DCT values and the IMC module is used to compute the inverse motion compensation. After the IMC the video frame contains all macro blocks in the frequency domain

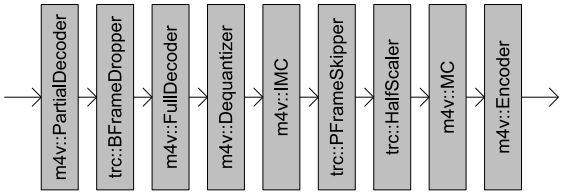

Figure 3: Transcoder Chain

without any temporal dependencies. The PFrameSkipper can skip P-frames now and the Scaler can reduce the spatial resolution of the frames. As mentioned before, both of them has to redefine the motion vectors of the remaining frames. The MC component implements the drift free loop to avoid drift errors and computes the motion compensation. There is no separate requantization transcoder in the chain, because requantization is already integrated in the drift free loop of the MC module. After the motion compensation has been computed, the frames are encoded into a video stream by the Encoder.

4.3 Implementation

We have created a prototype implementation of our proposed multidimensional transcoder, written in C++. This implementation is intended to be used for MPEG-4 video adaptation for mobile devices. For bit stream parsing and decoding of the video frames we have created a namespace called `m4v`, which contains all MPEG-4 specific classes. Classes implementing transcoding techniques are placed in a separate namespace called `trc`.

The frame-skipping and scaling transcoders are implemented in a way that they are independent of the video format which they process. Thus, they can also be used for other video coding formats, such as MPEG-1/2 or H261/3. To support such video formats, our implementation can be extended easily by adding new decoder and encoder modules which are capable of parsing the entire video streams.

The PFrameSkipper skips a frame whenever the controller requests this. It uses the forward dominant vector selection presented in [14] to find new motion vectors of following frames. The HalfScaler uses bilinear interpolation in the frequency domain as presented in [4]. In our current implementation we are able to downscale the spatial resolution by a factor of two. Thus, we have to compute one downscaled macro block from four incoming macro blocks. If each incoming macro block has only one motion vector, we can simply use those four vectors, each downscaled by a factor of two, in the downscaled macro block. Otherwise, if at least one of the incoming macro blocks has already four motion vectors, we use the vector of the most active block. The activity of a block is computed by the sum of all DCT coefficients. The module MC computes the motion compensation by using the redefined motion vectors and may use the original or a new quantization factor, depending on the request of the controller.

For testing purposes of our transcoders, we have also created a test application, which is a kind of a frontend to our transcoding architecture implementation. This application, which is written in Java, can be used to load a reference video in rawvideo format as well as a source MPEG-4 video. Furthermore a transcoding chain can be chosen to be used for transcoding. When starting the test, the source video is transcoded by the chosen transcoding chain and all three

Figure 4: Screenshot Test Application

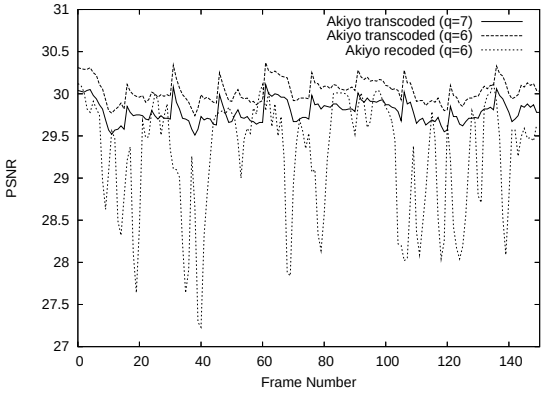

Figure 5: Akiyo, transcoded vs. recoded

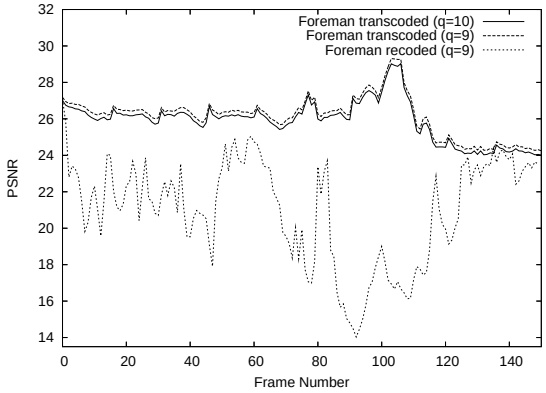

Figure 6: Foreman, transcoded vs. recoded

videos (reference, source and transcoded) are displayed on the screen. Additionally, the PSNR values of each frame are calculated and displayed in a graph. A screenshot of this application can be found in figure 4.

5. EVALUATION

For the evaluation of our implementation we selected two different videos. The first video, which is titled "Akiyo", shows a woman presenting some news. The background is quite static and there is not much motion in this video. The second one is called "Foreman" and there is more motion in the video. A foreman is talking into the camera and is gesticulating with his hands. In the last third of this video, the camera turns to the right and the shell of a building in the background becomes visible. Both videos have 300 frames in CIF (352×288) resolution, a group of video object panes (GOV) length of 30 and a frame rate of 25 fps. They were encoded to MPEG-4 by the use of the XviD codec and a fixed quantization level of one ($q = 1$). The Akiyo sequence has a bit rate of 512 KBit/s and the Foreman sequence a bit rate of 1024 KBit/s.

For both test videos we evaluated the PSNR values of the transcoded version compared to the recoded version of the sequence. The temporal as well as the spatial resolution were halved (QCIF, 12.5 fps) and the quantization factor was increased. The recoded version was produced by the use of the XviD codec and the transcoded version by the use of our implementation. The PSNR values were computed by upscaling each decompressed frame of the transcoded and recoded video respectively and using the uncompressed CIF frames of each video for reference. Figures 5 and 6 show the results of the transcoded video compared to the recoded one for both the Akiyo and the Foreman sequence. In both cases the PSNR values of the transcoded versions are higher than those of the recoded version. Because of the medium amount of motion in the Foreman sequence, the transcoded version benefits from the reuse of existing motion vectors and thus the PSNR differences between the recoded and the transcoded version are significantly higher than those of the Akiyo sequence. For the transcoded videos two different quantization factors are shown, because the bit rate of the transcoded version is a little bit higher than that of the recoded video when using the same quantization factor. Thus to compare the quality of the videos with the same bit rate we included the PSNR values of the transcoded video with a higher quantization factor in the figures. Additionally, we evaluated the achieved quality of our transcoder for the complete range of allowed quantization factors ($q = 2 - 31$). In figure 7 the average PSNR values of transcoded and recoded versions of the videos are shown. For lower bit rates our transcoder produces better or comparable PSNR values for both videos to the XviD codec. There are two outliers in

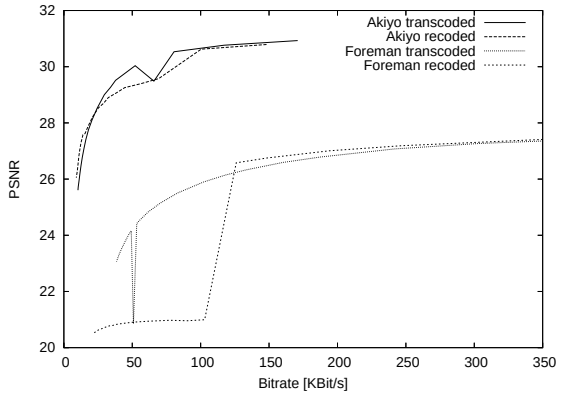

Figure 7: Average PSNR values

the graphs of the transcoded video: at 50 KBit/s ($q = 20$) for the Foreman sequence and at 65 KBit/s ($q = 5$) for the Akiyo sequence. However, it is not clear whether this is a bug of our implementation or is caused by the static frame dropping we are currently using. The runtime performance as well as the produced frame rate of our implementation, which was measured on an Intel Centrino platform with 1.7 GHz and 1GB RAM, are shown in table 2. Currently, only one of the video sequences can be transcoded in real-time on this platform. However, we have not yet optimized our implementation so that a comparison with existing codecs which are highly optimized is not suitable.

Sequence	Processing Time	Frame Rate
Akiyo	7.092 s	21.15 fps
Foreman	16.273 s	9.22 fps

Table 2: Transcoding Runtime Performance

6. CONCLUSION & FUTURE WORK

In this paper we have presented a new and flexible processing architecture for multidimensional video transcoding. We have identified three transcoding dimensions which are necessary for fine grain adaptation for mobile devices. Based on our architecture we have developed an integrated transcoder which transcodes an incoming video stream within the aforementioned three dimensions. The evaluation of our implementation shows promising results, with slight improvements of the produced quality with regards to PSNR values of the video frames. At the moment our implementation is not optimized as other video codecs on the market. Thus, the processing time needed for transcoding is quite high. But the evaluation results of the achieved quality are promising and we will optimize our implementation in the future. Additionally, by using advanced MPEG-4 features we can reduce the produced bit rate without affecting the quality of the stream and thus produce better quality results also for higher bit rates. Further, we plan to integrate an intelligent rate control mechanism into the controller in order to control the quality and bit rate of the transcoded video in the best possible way. Based on our transcoder implementation we further plan to evaluate the produced quality not only in terms of PSNR values but also by the use of subjective tests with potential users.

7. REFERENCES

[1] P. A. Assuncao and M. Ghanbari. A Frequency-Domain Video Transcoder for Dynamic Bit-Rate Reduction of MPEG-2 Bit Streams. *IEEE Transactions on Circuits and Systems for Video Technology*, 8(8):953–967, 1998.

[2] J. Brandt and L. Wolf. A Gateway Architecture for Mobile Multimedia Streaming. In *European Symposium on Mobile Media Delivery (EuMob06)*, Alghero, Italy, Sept. 2006.

[3] S.-F. Chang and D. Messerschmitt. Manipulation and compositing of MC-DCT compressed video. *IEEE Journal on Selected Areas in Communications*, 13(1):1–11, Jan. 1995.

[4] R. Dugad and N. Ahuja. A fast scheme for image size change in the compressed domain. *IEEE Transactions on Circuits and Systems for Video Technology*, 11(4):461–474, April 2001.

[5] K.-T. Fung and W.-C. Siu. DCT-based video frame-skipping transcoder. In *Proceedings of the 2003 International Symposium on Circuits and Systems, ISCAS '03*, volume 2, pages 656–659, May 2003.

[6] K.-T. Fung and W.-C. Siu. DCT-based video downscaling transcoder using split and merge technique. *IEEE Transactions on Image Processing*, 15(2):394 – 403, Feb. 2006.

[7] T. C. Huifang Sun, Xeumin Chen. *Digital Video Transcoding for Transmission and Storage*. CRC Press, 2005.

[8] Y.-R. Lee, C.-W. Lin, and C.-C. Kao. A DCT-Domain Video Transcoder for Spatial Resolution Downconversion. In *Proceedings of the 5th International Conference on Recent Advances in Visual Information Systems*, pages 207–218, London, UK, 2002. Springer-Verlag.

[9] S. Liu and A. Bovik. Local bandwidth constrained fast inverse motion compensation for DCT-domain video transcoding. *IEEE Transactions on Circuits and Systems for Video Technology*, 12(5):309–319, May 2002.

[10] H. S. S. Park, Y. Lee. Quality-adaptive requantization for low-energy MPEG-4 video decoding in mobile devices. *IEEE Transactions on Consumer Electronics*, 51(3):999–1005, 2005.

[11] T. Shanableh and M. Ghanbari. Heterogeneous video transcoding to lower spatio-temporal resolutions and different encoding formats. *IEEE Transactions on Multimedia*, 2(2):101–110, June 2000.

[12] J. Song and B.-L. Yeo. A fast algorithm for DCT-domain inverse motion compensation based on shared information in a macroblock. *Circuits and Systems for Video Technology, IEEE Transactions on*, 10(5):767–775, Aug. 2000.

[13] A. Vetro, C. Christopoulos, and H. Sun. Video transcoding architectures and techniques: an overview. *IEEE Signal Processing Magazine*, 20(2):18–29, Mar. 2003.

[14] J. Youn, M.-T. Sun, and C.-W. Lin. Motion vector refinement for high-performance transcoding. *IEEE Transactions on Multimedia*, 1(1):30–40, March 1999.