SPIE International Symposium on Voice, Video, and Data Communications, Boston, MA, November, 1998.

Compressed-Domain Reverse Play of MPEG Video Streams

Susie J. Wee[1] and Bhaskaran Vasudev[2]

[1]Hewlett-Packard Laboratories, Palo Alto, CA
[2]Epson Palo Alto Laboratory, Palo Alto, CA

ABSTRACT

We present several compressed-domain methods for reverse-play transcoding of MPEG video streams. A reverse-play transcoder takes any original MPEG IPB bitstream as input and creates an output MPEG IPB bitstream which, when decoded by a generic MPEG decoder, displays the original video frames in reverse order. A baseline spatial-domain method requires decoding the MPEG bitstream, storing and reordering the decoded video frames, and re-encoding the reordered video. The proposed compressed-domain transcoding methods achieve an order of magnitude reduction in computational complexity over the baseline spatial-domain approach. Much of the savings are achieved by using the forward motion vector fields available in the forward-play MPEG bitstream to efficiently generate the reverse motion vector fields used in the reverse-play MPEG bitstream. Furthermore, the storage requirements of the compressed-domain methods are reduced and the resulting image quality is within 0.6 dB of the baseline spatial-domain approach for a difficult highly detailed computer-generated video sequence. For more typical video sequences, the resulting image quality is even closer to the baseline spatial-domain approach.

Keywords: Reverse-play, MPEG, Compressed-domain processing, Video compression, Video processing, Transcoding.

1. INTRODUCTION

The advent of video compression standards such as MPEG-1 and MPEG-2 is fuelling the emergence of a wide range of devices and applications that exploit the compressed-domain representation of digital video, e.g. direct-broadcast by satellite services, DVD players, digital VCRs, digital TV broadcasts (DTV). These compression standards were however developed primarily for broadcast applications. In order to complete the transition to digital video from its current analog state, MPEG technology needs to encompass not just a compression methodology but also a video-processing framework. This will allow MPEG to be usable not just for the purposes of efficient storage and transmission of digital video, but also for systems wherein the user needs to interact with the digital video.

The predictive processing techniques employed in MPEG severely complicate typical video-processing functions that may need to be performed on MPEG-compressed bitstreams; such functions include rewind, fast-forward, reverse-play, splicing, video resizing, and frame deletion. Consider the task of deleting a single video frame from a video sequence. If the video were available in analog form, this task is extremely simple: drop the data for the deleted frame and modify the time-codes of the remaining frames. If however the video were available as an MPEG bitstream, the naïve approach would be to decode the bitstream, delete the desired frame, and re-encode the remaining frames. We refer to this as the baseline spatial-domain approach. This requires excessive storage and has significant computational complexity. Hence, in recent years, there has been a great deal of research in developing efficient techniques for manipulating digital video when it is available only in compressed form; we refer to these techniques as compressed-domain processing methods. Compressed-domain processing may offer several advantages vis-à-vis' spatial-domain processing, such as (a) smaller data volume, (b) lower computational complexity since the complete decode and re-encode cycle may be avoided, and (c) preservation of image fidelity since the cascading of compression processes can often be eliminated. Several compressed domain processing methods for video resizing, inverse motion compensation, filtering, and splicing have been developed in [1]-[4]. The algorithms developed in [1]-[3] indicate computational savings when the processing is performed directly on intermediate DCT-domain data, while the algorithm developed in [4] achieves computational savings by exploiting the prediction structure used in MPEG.

In this paper, we develop a compressed-domain approach for the following scenario: given a forward-coded MPEG bitstream, our goal is to create a new MPEG bitstream that when decoded displays the video frames in the reverse order from the original MPEG bitstream. Note that the reverse-play function is desirable in many applications, e.g. DVD players, digital

VCRs, and digital video servers. For uncompressed video the solution for reverse-play is simple: reorder the video frame data in reverse order. The simplicity of this solution relies on two properties: the data for each video frame is self-contained and it is independent of its placement in the data stream. These properties typically do not hold true for MPEG-coded video data. Reverse-play transcoding is difficult because MPEG compression uses predictive processing techniques that are not invariant to changes in frame order e.g. simply reversing the order of the input frame data will not reverse the order of the decoded video frames. Furthermore, reversing the order of the input video frames requires the generation of a “reversed” motion vector field. We show that if the transcoding is performed carefully, much of the motion vector information contained in the original MPEG video stream can be reused to efficiently generate the "reversed" motion vector field.

We have developed a reverse-play transcoding algorithm that operates directly on the compressed-domain data and can handle all combinations of intraframe (I), predicted (P), and bidirectionally predicted (B) coded video frames. Unlike the approaches in [1]-[3], we do not manipulate the DCT data to realize computational savings compared to the spatial-domain approach. Instead, the computational savings in our approach come from the reuse of the motion vector fields contained in the input MPEG bitstream. The proposed algorithm is simple and achieves high performance with low computational complexity and low memory requirements.

This paper begins with a brief description of some aspects of the MPEG video compression algorithm that are relevant to this work. The transcoding problem is then defined and several architectures are developed for compressed-domain reverse-play transcoding. These architectures take any MPEG coded bitstream as input and generate an output standard-compliant MPEG bitstream which, when decoded with a generic MPEG decoder, yields video playback in the reverse order from which it was originally acquired. Simulations of the proposed transcoding schemes indicate that high quality reverse-play MPEG bitstreams can be generated with less than 0.6dB loss in image quality and with significantly lower computational complexity than the baseline spatial-domain approach. This technique can be implemented in digital VCRs, video servers, and any other devices requiring smooth frame-by-frame reverse-play functionality with minimal increase in storage.

2. BACKGROUND

The proposed compressed-domain, reverse-play transcoding algorithms were designed to exploit various features of the MPEG video compression standard. Detailed descriptions of the MPEG video compression standard can be found in [5]-[7]. In this section, we briefly discuss some aspects of the standard that are relevant to this work.

In MPEG, each frame in a video sequence is coded with one of three modes: intraframe (I), forward prediction (P), and bidirectional prediction (B). A typical IPB pattern in *display order* and *coding order* is shown in Figure 1.

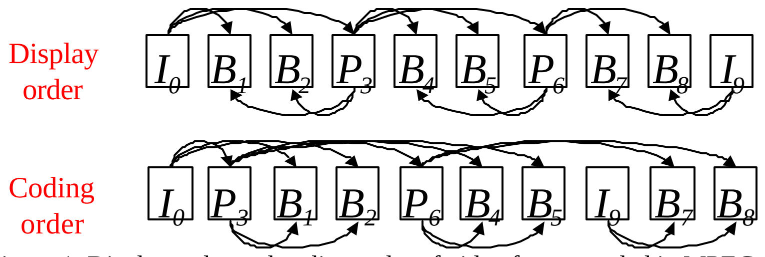
Display order
I_0 B_1 B_2 P_3 B_4 B_5 P_6 B_7 B_8 I_9
Coding order
I_0 P_3 B_1 B_2 P_6 B_4 B_5 I_9 B_7 B_8

Figure 1: Display order and coding order of video frames coded in MPEG.

I frames are coded independently of other frames. P frames depend on a prediction based on a previously coded I or P frame. B frames depend on a prediction based on the preceding and following I or P frames. Notice that each B frame depends on data from a past and future frame, i.e. a future frame must be (de)coded before the current B frame can be (de)coded. For this reason, the coding order is distinguished from the display order. For each frame coded as a P or B frame, MPEG uses block motion-compensated prediction to reduce the temporal redundancies inherent to video. For example, MPEG exploits temporal redundancies between 16x16 macroblocks in frame P_3 and a corresponding 16x16 square region in frame I_0. The correspondence is then represented using a motion vector (MV) for the 16x16 block in P_3. Since the correspondences may not be exact, the residuals are coded using an 8x8 discrete cosine transform (DCT).

3. COMPRESSED-DOMAIN PROCESSING

The goal of transcoding is to perform video-processing operations on standard-compliant compressed bitstreams. In MPEG transcoding, both the input and output data are MPEG-compliant video streams. The baseline approach for performing a transcoding operation is shown in the upper path of Figure 2. First, the MPEG video stream is completely decompressed into its pixel-domain representation; the pixel-domain video is then stored and processed; and finally the processed video is re-encoded into a new MPEG video stream. This baseline approach can be computationally expensive and can have large memory requirements. In addition, the quality of the coded video can deteriorate with each recoding cycle.

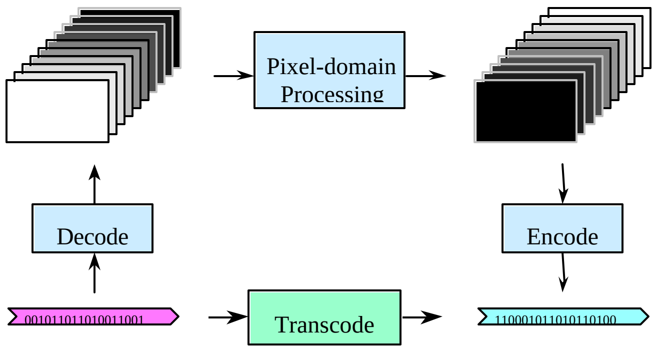

Figure 2: Spatial-domain (baseline) versus compressed-domain approach to video processing of MPEG bitstreams.

For some transcoding operations, it may be possible to design more efficient solutions by exploiting the properties of the compression algorithm and the desired video-processing operation. This is referred to as the compressed-domain approach and is performed by the Transcode module shown in Figure 2. For example, improved efficiency may be achieved by only partially decompressing the video stream and performing the processing directly on the compressed-domain data. In addition, further gains may be achieved by selectively processing the portions of the video that are affected by the video processing operation.

In this paper, we address the problem of performing a reverse-play transcoding operation on MPEG-compressed video streams. In other words, given an MPEG stream as input, we wish to generate an output MPEG stream that emulates reverse play in a generic MPEG decoder. Rather than taking the baseline approach of decoding the MPEG stream, storing and reordering the decoded frames, and re-encoding the result, we take the compressed-domain approach and develop more computation- and memory-efficient solutions. These solutions exploit the properties of the MPEG video compression standard and the reverse-play operation.

Specifically, in MPEG encoding, 60-80% of the computational complexity of the encoder is in the estimation of the motion vector fields. In our work, we reduce the computational complexity for reverse-play transcoding by processing the MV fields that are obtained from the input MPEG bitstream; thus, the motion-estimation task is eliminated during transcoding. Unlike the compressed-domain work of [1]-[3], in our work, we decompress the DCT data since the savings obtained using DCT-domain processing is not significant compared with the savings realized by processing the motion vector fields. In future work, we will report on algorithms for reverse-play that avoid the Inverse DCT-DCT conversion steps.

In the next section, we describe in detail various approaches that can be adopted for the Transcode module in Figure 2 for generating MPEG bitstreams suitable for reverse play.

4. COMPRESSED-DOMAIN REVERSE-PLAY TRANSCODING ALGORITHMS

Several approaches can be adopted for enabling the reverse play of an original (forward-play) MPEG bitstream. In this section, we examine several architectures for reverse-play transcoding. A reverse-play transcoder takes the original MPEG bitstream as input and creates as output a new MPEG stream which, when decoded by a generic MPEG decoder, plays the original video frames in reverse order.

Reverse-Play Transcoding Architecture #1 – Baseline Approach

The first architecture shown in Figure 3 describes the baseline approach in which the input MPEG video stream is decoded into uncompressed video frames, i.e. its pixel-domain representation. These frames are stored and reordered in a frame buffer, and the result is re-encoded with an MPEG encoder. The GOP structure used by MPEG allows this processing to be performed on a GOP-by-GOP basis; a typical IPB GOP may contain 15 frames. A difficulty that arises is that this transcoder requires a frame buffer that stores the frames of the entire GOP; in this case, the frame buffer must be large enough to store 15 frames. The re-encoding process causes two additional difficulties. First, motion estimation is a computationally expensive operation that accounts for well over half of the computations needed for the entire transcoder. Second, the decoupled decoder and encoder cause a generation loss in the quality of the reconstructed video frames.

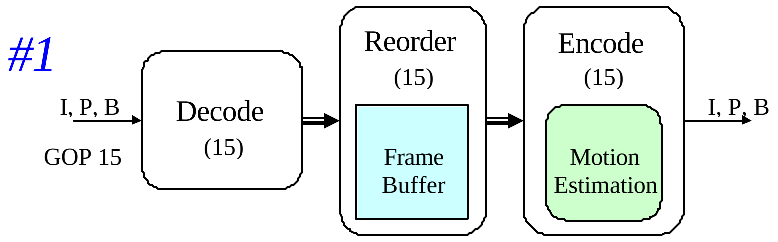

Figure 3: Architecture #1 for reverse-play transcoding.

Besides the baseline pixel-domain approach, one can envision an analogous baseline DCT-domain scheme, which we will call an I-frame only scheme. This is identical to the baseline approach since it requires decoding the DCT-domain IPB-frame data to DCT-domain I-frame data followed by re-encoding to an IPB MPEG bitstream. DCT-domain motion compensation and motion estimation modules may be used in this approach.

In practice, both the baseline pixel- and DCT-domain approaches are not feasible due to the excessive computational requirements needed by the full MPEG decode and re-encode processes and/or the storage needed for the decoded video data. We will now examine architectures that have lower storage requirements and may not need the full MPEG decoding and re-encoding cycle.

Reverse-Play Transcoding Architecture #2

The second reverse-play transcoder improves the efficiency of the baseline approach by 1) lowering the storage requirements of the transcoder and 2) reducing the necessity of the full MPEG decode and re-encode cycle. This is achieved by exploiting the symmetry of the B-frame prediction process used in MPEG video compression. Specifically, each B frame is coded with a predicted and a residual component. The predicted component is based on two anchor frames, one that proceeds and one that follows the actual B frame (in display order). A backward and a forward motion vector field describe the prediction from the proceeding and following anchor frames. The goal of the reverse-play operation is to reverse the order of the frames. In the coded stream, this will result in a swapping of the forward and backward anchor frames. To compensate for this reordering of the anchor frames, the B frames can simply swap the backward and forward motion vectors [8]. This B-frame swapping trick results in significant computational and memory savings in the reverse-play transcoder. The P-frame prediction process does not have this type of forward/backward symmetry; thus, P frames must be processed appropriately.

The architecture of the second reverse-play transcoder is shown in Figure 4. The input MPEG bitstream is first separated into two parts; the first contains the portions of the bitstream representing the I and P frames and the second contains the portions of the bitstream representing the B frames. This separation process can be accomplished by simply searching the MPEG bitstream for picture startcodes and parsing the first few bytes of the picture header to determine the picture type. As in the baseline architecture, the I and P frames are decoded, the resulting frames are stored and reordered, and the reordered frames are re-encoded into an I or IP MPEG bitstream. The B frames, however, only need to be Huffman decoded and re-encoded to extract and swap the forward and back motion vectors and coding modes. The processed data is then combined into a compliant MPEG bitstream. The frame storage and full decoding and re-encoding requirements are reduced to the number of I and P frames in each GOP, which is 5 frames for a typical 15-frame IPB GOP. An issue that remains is to determine how to best process the I and P frames.

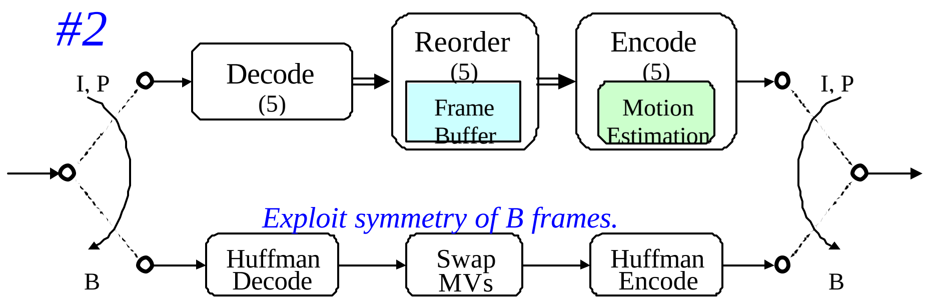

Figure 4: Architecture #2 for reverse-play transcoding.

Reverse-Play Transcoding Architecture #3

A third reverse-play transcoding architecture is shown in Figure 5. As with architecture #2, this architecture improves the efficiency of the baseline approach by 1) lowering the storage requirements of the transcoder and 2) reducing the necessity of the full MPEG decode and re-encode cycle. In addition, it improves the efficiency of the second architecture by reducing the computational requirements needed for motion estimation. Specifically, in architecture #2, the motion vectors used during the re-encoding process are derived by performing motion estimation on the reordered frames. In this architecture, the motion vectors are derived more efficiently by exploiting the motion vector information given in the input MPEG video stream. The remaining task is to find an efficient and effective method of estimating the reverse motion vectors from the forward motion vectors [9]. This is the topic of Section 5.

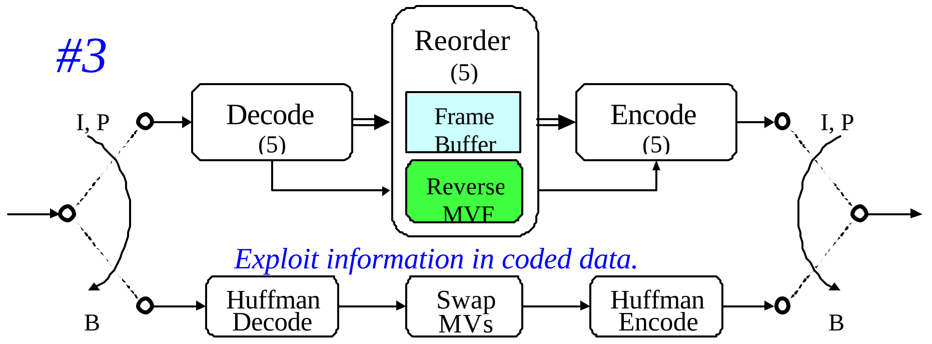

Figure 5: Architecture #3 for reverse-play transcoding.

Reverse-Play Transcoding Algorithms

References [8] and [10] address the problem of reverse-play of MPEG video streams. While they do not specifically address the transcoding problem, the ideas presented in their work can be extended to fit the problem. In [10], a method is described for transforming an MPEG data stream into a local compressed form using a P-to-I frame conversion. During forward play in a video player device, after each P frame is retrieved, decompressed, and played out, it is encoded as an I frame and stored in local storage. The forward-play sequence I_0 P_3 B_1 B_2 P_6 B_4 B_5 I_9 B_7 B_8 (in coding order) is converted to an IB frame sequence I_0 I_3 B_1 B_2 I_6 B_4 B_5 I_9 B_7 B_8, where I_3 and I_6 are converted P frames. For reverse playback, the frames are reversed before they are input to the decoder; during this reversal process, the locally stored I_3 and I_6 frames replace frames P_3 and P_6. This basic idea can be extended to the reverse-play transcoding algorithm shown in Figure 6. Note that once the IPB frame syntax has been changed to the IB frame syntax, the approach developed in [8] can be used for reverse-play.

The P-to-I frame conversion can be done in the DCT domain [1]-[2] or in the pixel domain by simply decompressing the P macroblocks to and performing the motion compensation in the DCT or pixel domain. If frame stores are available for decompression, the spatial-domain approach has lower computational complexity than the DCT-domain approaches.

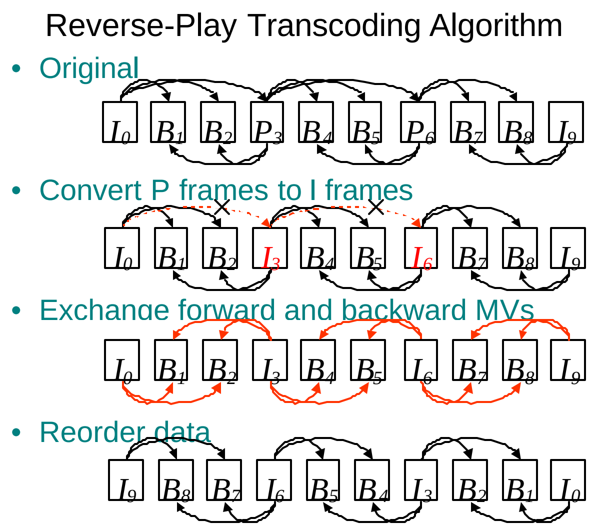
Reverse-Play Transcoding Algorithm
Original
I_0 B_1 B_2 P_3 B_4 B_5 P_6 B_7 B_8 I_9
Convert P frames to I frames
I_0 B_1 B_2 I_3 B_4 B_5 I_6 B_7 B_8 I_9
Exchange forward and backward MVs
I_0 B_1 B_2 I_3 B_4 B_5 I_6 B_7 B_8 I_9
Reorder data
I_9 B_8 B_7 I_6 B_5 B_4 I_3 B_2 B_1 I_0

Figure 6: IPB to IB frame conversions for compressed-domain reverse-play transcoding.

The approaches proposed in [8] and [10] are however quite restrictive in that they only address reverse playback of MPEG bitstreams with the IB syntax. The IB syntax may be used in a digital studio setting; however, most MPEG coded video for broadcast applications will conform to the IPB syntax. Compared to I frames, P frames need only 1/2 as much storage and B frames need only 1/4th the storage [11]. For the group of frames shown in Figure 6, the storage requirements for the scheme proposed in [10] is 20% more than that needed for the original MPEG coded sequence. Since generic MPEG decoders may not have this additional storage, this scheme may not be readily usable in conventional MPEG decoding systems.

The reverse-play transcoding algorithm shown in Figure 6 converts the MPEG IPB bitstream to an MPEG IB bitstream. The drawback of this approach is the increased bit rate that results from the P-to-I frame conversions. In this work, we propose a reverse-play transcoding algorithm that overcomes this drawback; this algorithm is shown in Figure 7. This algorithm differs from the first in that rather than performing P-to-I frame conversions, the I and P frames are converted to a new set of reverse I and P frames. The reverse IP frames can be coded to the same bit rate as the original IP frames, thus resulting in an output MPEG stream that has the same bit rate as the input. This algorithm can be implemented with the three reverse-play transcoding architectures discussed earlier. The computational requirements of architectures #2 and #3 are significantly reduced from the baseline approach of architecture #1 due to the savings in processing the B frames. Architecture #3 achieves additional savings over architecture #2 by significantly reducing the computations needed for motion estimation during the re-encoding process.

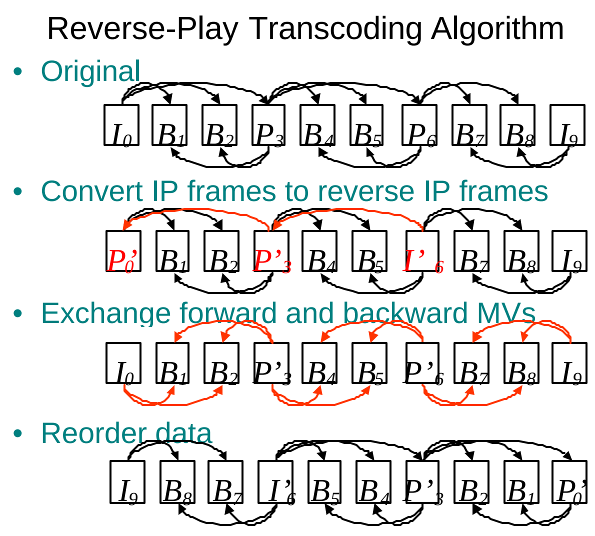
Reverse-Play Transcoding Algorithm
• Original
I0 B1 B2 P3 B4 B5 P6 B7 B8 I9
• Convert IP frames to reverse IP frames
P0' B1 B2 P'3 B4 B5 I'6 B7 B8 I9
• Exchange forward and backward MVs
I0 B1 B2 P'3 B4 B5 P'6 B7 B8 I9
• Reorder data
I9 B8 B7 I'6 B5 B4 P'3 B2 B1 P0'

Figure 7: IPB to IPB' frame conversions for compressed-domain reverse-play transcoding.

5. REVERSING MOTION VECTOR FIELDS

In the previous section, we described three architectures for reverse-play transcoding of MPEG video streams. The first architecture requires all the video frames to undergo a full decode and re-encode cycle. The second and third architectures only require the I and P frames to undergo a full decode and re-encode cycle. In architecture #2, the motion vectors used in the re-encoding process are computed by performing motion estimation on the decompressed, reordered video frames. In this architecture, the motion estimation process is completely decoupled from the input data stream and its decoding process. In architecture #3, it is suggested that improved computational efficiency can be gained by estimating the reverse motion vector field from the motion vector information provided by the input data stream.

The problem of estimating a reverse motion vector field from a given forward motion vector field was examined in [9]. A number of methods were presented that tradeoff motion vector accuracy for computational efficiency. The performance of each method was evaluated by examining the PSNR of the prediction that resulted when using the estimated reverse motion vector field. In this work, we examine the performance of the two simplest methods, in-place reversal and maximum overlap, in our reverse-play transcoding architecture. Each of these methods is discussed below.

We begin by briefly describing the problem that arises in the reverse-play transcoding algorithm shown in Figure 6. The forward motion vector field for P_6 is given in the input data stream and used in the decoding portion of the reverse-play transcoder. This forward motion vector field describes the motion between 16x16 macroblocks in frame P_6 relative to 16x16 blocks in frame P_3. Given this forward motion vector field $MVF_{A,B}$, our objective is to estimate the reverse motion vector field $MVF_{B,A}$ which describes the motion between 16x16 macroblocks in frame P_3 relative to 16x16 blocks in frame P_6. The problem of estimating the reverse motion vector field becomes quite difficult since only sparse motion vector fields are available, i.e. in MPEG the motion vectors are only available on a 16x16 sampling grid. This is illustrated with a simple example shown in Figure 8. The top half of Figure 8 shows the forward motion vector for a single macroblock in P_6. This motion vector specifies the 16x16 block in frame P_3 that is used as a prediction for the current macroblock in P_6. The problem that now remains is to compute the motion vector for the corresponding macroblock in P_3 in relation to a 16x16 block in P_6.

In-place Reversal
The in-place reversal method simply requires reversing each element of the forward motion vector field such that $MVF_{B,A} = -MVF_{A,B}$. This is illustrated in the lower half of Figure 8. The in-place reversal method performs well in the interiors of objects undergoing uniform translational motion. However, in the case of larger motions, it often produces incorrect motion vectors near the object boundaries. These incorrect motion vectors create false associations between unrelated blocks in the two video frames and thus can result in poor performance. Note that the in-place reversal method only requires one operation per macroblock to generate the reverse motion vector for that macroblock.

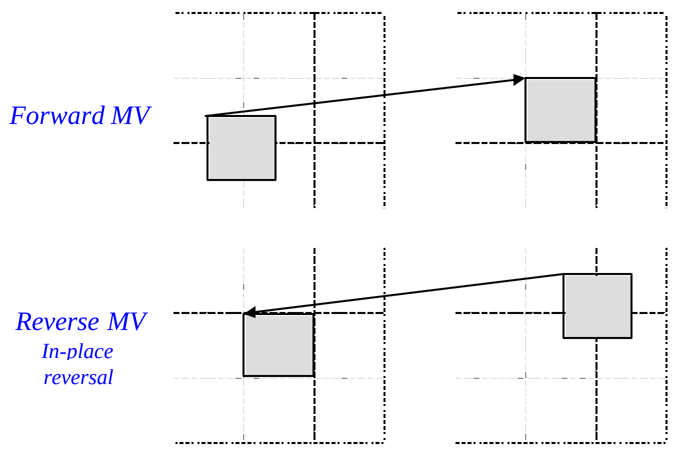

Figure 8: Reversing motion vector fields with the in-place reversal method.

Maximum-Overlap
A better estimate of the reverse motion vector field can be obtained by using a local neighborhood of motion vectors from the forward motion vector field. Consider a neighborhood of macroblocks and their corresponding forward motion vectors as shown in left half of Figure 9; here, we show the local nine-block neighborhood including and surrounding the current macroblock. Each forward motion vector in the neighborhood can be assigned a value that represents its importance or relevance in relation to the macroblock for which we must estimate the reverse motion vector. This importance is quantified by a weighting function w(k,l), where k and l represent the local index of the neighborhood of motion vectors.

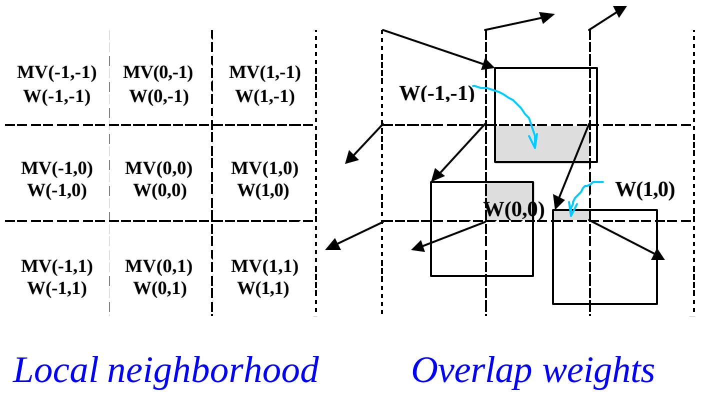

Figure 9: Local neighborhood and weights.

The maximum-overlap method considers the corresponding forward motion vector and its neighbors. The motion vector with the largest weight, i.e. the one that overlaps the block with the largest area, is negated and used as the estimate of the reverse MV. In Figure 9, for example, w(-1,-1) has the largest weight; hence, for this macroblock, the reverse motion vector is simply -MV(-1,-1). If on the other hand w(1,0) had the largest weight, then the estimated reverse motion vector would have been -MV(1,0).

For small translational motions, the maximum-overlap and in-place reversal methods yield the same estimates of the reverse motion vectors. However, for larger translations, the maximum-overlap method produces better estimates. From a complexity viewpoint, the area of overlap must be computed for each motion vector in the neighborhood and the region with the maximum overlap must be chosen. Each weight can be computed with 6 adds and a multiply. For the nine-block neighborhood, finding the region with the maximum overlap and negating its corresponding motion vector requires 54 adds, 9 multiplies, 8 compares, and 1 negation for a total operation count of 72 operations per output motion vector. This is the worst case result. Note that for the in-place reversal scheme, only 1 operation is needed per output motion vector. A spatial-domain approach using a logarithmic search based motion estimation method [7] would require 19200 operations for a +/- 7 pixel search range; furthermore, unlike our compressed-domain approach, additional operations would be needed for sub-pixel accurate motion vectors. A detailed discussion of reverse motion vector fields can be found in [9].

Here, we summarize the processing and storage requirements for the various transcoding schemes discussed in this paper. Since motion estimation dominates the computational cost in an MPEG encoder, our estimate of processing requirements is for the motion-estimation portion only.

1. Baseline Approach: Besides Huffman decoding, inverse DCT, motion compensation, motion compensated prediction, DCT and Huffman encoding, we have to perform motion estimation for the P and B frames in the output bitstream. Assuming a logarithmic search method for motion estimation and a +/- 7 pixel search range, the motion estimation cost per macroblock for integer-valued motion vectors is 19200 operations; additional operations are needed for subpixel valued motion vectors. Since motion estimation is nearly 80% of the computational cost of MPEG encoding, this task dominates over the other tasks needed for the baseline approach. Furthermore, the baseline approach requires a frame buffer that stores all the frames of the GOP.
2. Transcoding from IPB to IB: In this scheme full MPEG decoding is done only for the I and P frames. Motion estimation need not be performed for the B frames since their forward and backward motion vectors are simply swapped. For the example in Figure 5, this method requires a frame buffer that stores the I and P frames in the GOP.
3. Transcoding from IPB to IPB' using in-place reversal: As in the previous case, full MPEG decoding is done only for the I and P pictures. Modified P pictures are created using a simple in-place reversal of the forward motion vectors of original P pictures. This costs one operation per macroblock compared to the 19200 operations needed for the baseline approach. The storage requirements are same as in the previous case.
4. Transcoding from IPB to IPB' using maximum overlap: All of the steps are identical to the in-place reversal scheme except for the estimation of the reverse motion vector fields which requires 72 operations per macroblock. This cost is higher than that needed in the previous case; however, it is a factor of 260 lower than the motion estimation cost of the baseline approach. The storage requirements are same as in the previous case.

6. EXPERIMENTAL RESULTS

The proposed reverse-play transcoding architectures achieve an order of magnitude reduction in computational requirements as compared to the baseline approach by practically eliminating the number of computations needed to perform motion estimation during the re-encoding process. In this section, we examine the video quality of the resulting reverse-play MPEG streams. Experimental results show that the proposed architectures result in very little loss in PSNR. The details of the experiments are described below.

Video sequences were processed in sets of 46 frames. The sequences are listed in the table below. All sequences were downsampled and/or cropped to CIF resolution of 352x240 pixels/frame at 30 frames/sec. The girl sequence contains synthetically generated panning and zooming motions of a high-resolution original image. In addition, it contains a fixed high-resolution rotating test pattern. The first part of the sequence predominantly contains panning motions, and the last part of the sequence predominantly contains zooming motions of increasing speed. The bus, carousel, and football sequences are MPEG test sequences containing natural video scenes.

Number	Video Sequence	Frames	R_{IPB}, R_{IP} (Mbps)
1	Girl	0-45	2.75, 1.33
2	Bus	0-45	2.75, 1.33
3	Bus	45-90	2.75, 1.33
4	Bus	90-135	2.75, 1.33
5	Carousel	0-45	1.375, 0.67
6	Carousel	45-90	1.375, 0.67
7	Carousel	90-135	1.375, 0.67
8	Football	0-45	1.375, 0.67
9	Football	45-90	1.375, 0.67
10	Football	90-135	1.375, 0.67

The video sequences were first MPEG coded in forward order. The resulting MPEG stream was used as the input to a number of reverse-play transcoding methods, each of which is described below.

Forward
The original frames of each sequence were MPEG coded at rate R_{IPB}. The resulting MPEG stream is referred to as the forward-coded stream. The 46 frames were coded with three 15-frame IPB GOPs and one I frame. The PSNRs of the reconstructed frames are shown in each plot in reverse order. The forward-coded stream is the input to each of the reverse-play transcoding algorithms. Thus, the PSNR of its reconstructed frames provide an upper bound on the performance that can be achieved with the compressed-domain transcoding algorithms.

Reverse #1
The forward-coded stream is first decompressed to its reconstructed video frames. The ordering of these frames is reversed, and the resulting reverse-order frames are re-encoded with MPEG at rate R_{IPB} with three 15-frame IPB GOPs and one I frame. Motion estimation was performed with full-search block matching during the re-encoding process.

Reverse #2
The forward-coded stream is de-multiplexed into two streams: the picture data for the I and P frames make up the first stream and the picture data for the B frames make up the second stream. The first 15 of the 16 frames in the IP stream are then decompressed to its reconstructed video frames. The ordering of these frames is reversed, and the resulting reverse-order frames are re-encoded with MPEG at rate R_{IP} with 5-frame IP GOPs. Motion estimation was performed with full-search block matching during the re-encoding process. The B frames are simply Huffman decoded; their forward and backward coding modes and motion vectors are swapped on a macroblock-by-macroblock basis; and the result is Huffman re-encoded. The resulting IP and B streams are then combined appropriately to form the output MPEG bitstream.

Reverse #3a and #3b
The forward-coded stream is de-multiplexed into two streams: the picture data for the I and P frames make up the first stream and the picture data for the B frames make up the second stream. The first 15 of the 16 frames in the IP stream are then decompressed to its reconstructed video frames. The ordering of the reconstructed frames is reversed, and the resulting reverse-order frames are re-encoded with MPEG at rate R_{IP} with three 5-frame IP GOPs. The motion vectors from the forward coded stream are saved during the decoding process. These motion vectors are used to derive the motion vectors that are used in the following re-encoding process. Method #3a uses the maximum-overlap method and method #3b uses the in-place reversal method. The B frames are simply Huffman decoded; their forward and backward coding modes and motion vectors are swapped on a macroblock-by-macroblock basis; and the result is Huffman re-encoded. The resulting IP and B streams are then combined appropriately to form the output MPEG bitstream.

In summary, the original frames were first coded in forward order, creating the forward-coded MPEG stream. Reverse #1 decodes the forward-coded stream and re-encodes the frames in reverse order with 15-frame IPB GOPs and full-search motion estimation. Reverse #2 differs from Reverse #1 in that it only re-encodes the I and P frames, while for the B frames the forward and backward coding modes and motion vectors are swapped. The resulting I, P, and B frames are then appropriately combined into an MPEG stream. The re-encoding of the I and P frames uses full-search motion estimation. Reverse #3a and #3b differ from Reverse #2 in that rather than determining the reverse motion vectors with full-search motion estimation, they are determined with the maximum overlap and in-place reversal methods. These methods process the motion vectors that are given in the forward-coded stream.

The PSNR plots for each frame of sequences 1, 3, 5, and 9 are shown in Figure 10. These plots provide a good overview of the performance because the three bus sequences, the three carousel sequences, and the three football sequences performed similarly. The plot in Figure 11 shows the average PSNR performance across all the frames of each sequence. These average PSNRs are shown relative to the average PSNR of the forward-coded frames.

The girl sequence showed the largest PSNR loss in the transcoding process. This is due to the high texture and detail in this synthetically generated sequence. The full re-encoding process results in a PSNR loss of 1.9 dB. The computational savings gained by simplifying the B-frame processing results in a loss of another 0.34 dB. The further computational savings gained by using the maximum overlap method for motion estimation loses another 0.24 dB and the in-place reverse method loses another 0.07 dB. Notice that for the girl sequence, the difference in performance across the reverse-play transcoding algorithms is 0.65 dB. This quality loss may be acceptable for the significant reduction in computational requirements.

The bus sequences also showed measurable differences in performance for the various schemes. The overall drop in PSNR due to the transcoding operation is not as severe as the drop for the girl sequence. This is due to the lack of high texture in the sequence. These sequences exhibit less of a hit in performance when simplifying the B-frame processing. This indicates that much of the error is due to B-frame quantization, and not only due to the re-encoding of its anchor frames. For the three bus sequences, the maximum overlap method outperforms the in-place reversal method by 0.2, 0.1, and 0.13 dB. This is because the bus sequence has very large translational motions, which cause the in-place reversal method to fail at object boundaries where the maximum overlap method succeeds.

The carousel and football sequences result in less differentiation between the schemes, i.e. all the methods seem to perform similarly. The differences in PSNR are so small (often within 0.05 dB) that they may not be reliable measures for differentiation. This may in part be due to the lower bit rate coding, which sacrifices some of the finer detail in the scenes. When the details are washed out by the coding, the accuracy of the motion vectors becomes less important. In addition, the carousel sequence undergoes motions that are not well modeled by block-based motion estimation and compensation methods. The football sequence has added difficulty in the blurred source material due to the fast movements of the camera and the football players. The lessened importance of motion vector accuracy supports the use of simplified reverse-play transcoding algorithms in that in these cases, the computational benefits of the proposed algorithms result in no loss in quality of the reconstructed frames.

Experimental results show that the proposed reverse-play transcoding algorithms provide considerable reductions in computational requirements with little loss in PSNR performance. Specifically, the computations needed to perform motion estimation are practically eliminated, thus significantly reducing the overall computational requirements. This is typically achieved with little if any loss in PSNR. Highly textured and detailed sequences such as the synthetically generated girl sequence show a measurable difference in the various reverse-play transcoding algorithms. The various algorithms are within 0.6 dB in performance. The bus sequence provides variations of about 0.4 dB. The carousel and football sequences show little performance difference between the various algorithms. The maximum overlap method is recommended over the in-place reversal method because it provides better performance in scenes with large motions with little added complexity.

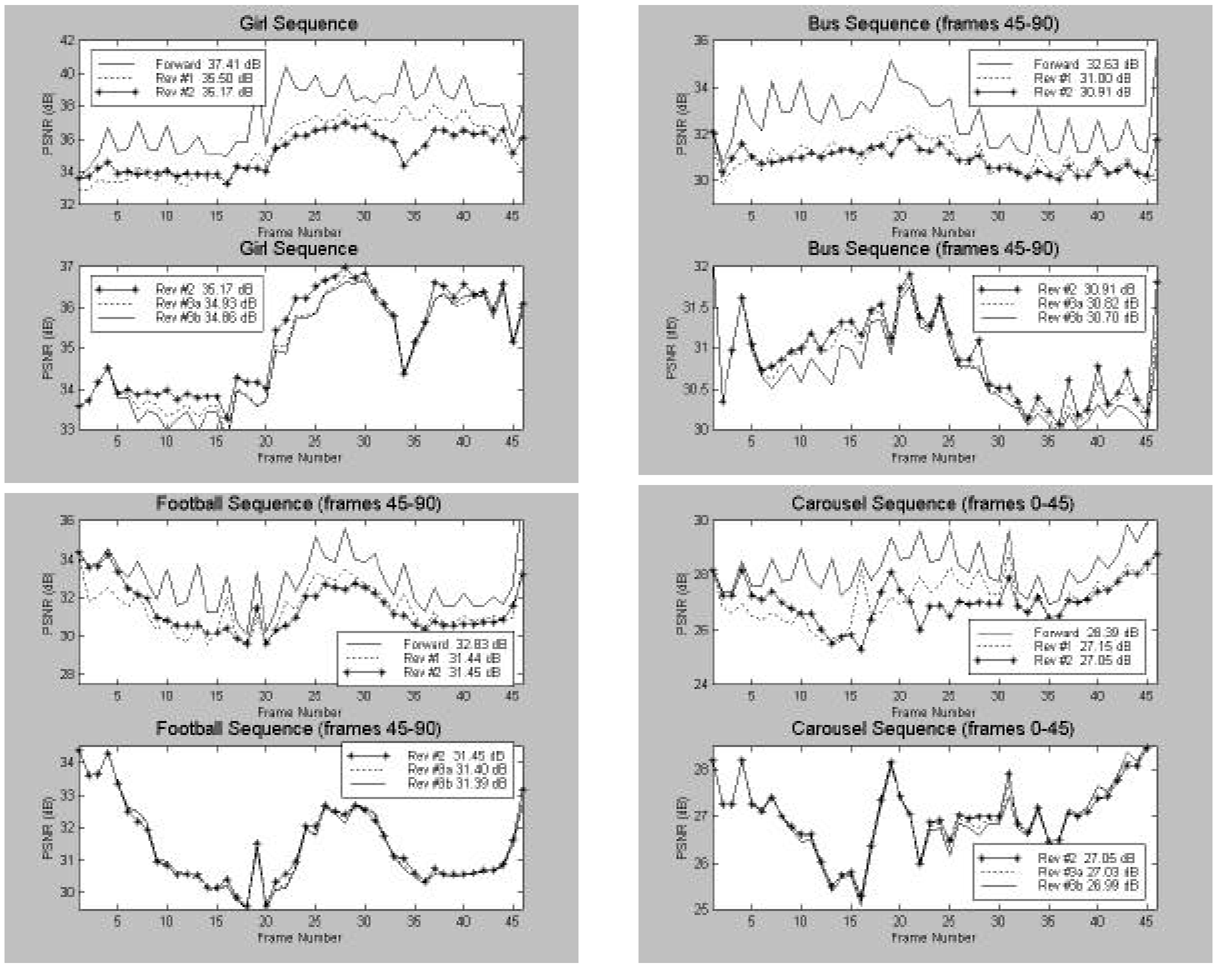

Figure 10: PSNR for various MPEG reverse-play transcoding schemes.

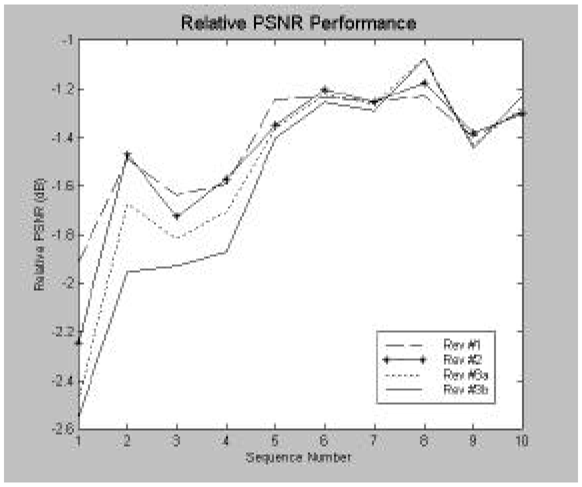

Figure 11: Average PSNR for various MPEG reverse-play transcoding schemes.

7. CONCLUDING REMARKS

Several simple transcoding schemes have been developed for the reverse play of forward-play MPEG bitstreams. Unlike baseline spatial-domain methods, which require full MPEG decoding and re-encoding to generate the reverse-play MPEG bitstreams, the proposed transcoders are compressed-domain schemes that exploit the properties of MPEG video compression and the reverse-play operation. The compressed-domain transcoding schemes developed here reuse the forward motion vector fields available in the forward-play MPEG bitstream to efficiently generate the reverse motion vector fields used in the reverse-play MPEG bitstream. These transcoders offer an order of magnitude reduction in computational complexity relative to the baseline spatial-domain approach by practically eliminating the number of computations needed for motion estimation; furthermore, the storage requirements are reduced relative to the spatial-domain approach. From an image quality viewpoint, PSNR degradations for the compressed-domain schemes are within 0.6 dB relative to the baseline spatial-domain approach for the same coded bit rate. Unlike the schemes in [8] and [10], the proposed transcoders are capable of generating IPB bitstreams for reverse-play at the same bit rate and a similar IPB frame structure as the forward-play MPEG bitstream.

8. REFERENCES

1. S.-F. Chang and D. Messerschmitt, "Manipulation and compositing of MC-DCT compressed video", IEEE Journal on Selected Areas in Communications, vol. 13, Jan. 1995.
2. N. Merhav and V. Bhaskaran, "Fast algorithms for DCT-domain image downsampling and for inverse motion compensation", IEEE Transactions on Circuits and Systems for Video Technology, vol. 7, June 1997.
3. N. Merhav and R. Kresch, "Approximate convolution using DCT coefficient multipliers", IEEE Transactions on Circuits and Systems for Video Technology, vol. 8, Aug. 1998.
4. S. Wee and V. Bhaskaran, "Splicing MPEG video streams in the compressed-domain", IEEE Workshop on Multimedia Signal Processing, June 1997.
5. MPEG-2 International Standard, Video Recommendation ITU-T H.262, ISO/IEC 13818-2, Jan. 1995.
6. J. Mitchell, W. Pennebaker, C. Fogg, and D. LeGall, "MPEG Video Compression Standard", Digital Multimedia Standards Series, Chapman and Hall, 1997.
7. V. Bhaskaran and K. Konstantinides, "Image and Video Compression Standards: Algorithms and Architectures", Kluwer Academic Publishers, Second Edition, June 1997.
8. S. Chen, "Reverse playback of MPEG video", U.S. Patent 5,739,862.
9. S. Wee, "Reversing motion vector fields", IEEE International Conference on Image Processing, Oct. 1998.
10. M.-S. Chen and D. Kandlur, "Downloading and stream conversion: supporting interactive playout of videos in a client station", Proceedings of International Conference on Multimedia Computing, May 1995.
11. A. Rodriguez and D. Fibush and S. Bilow, "MPEG-2 fundamentals for Broadcast and Post-production Engineers", Tektronix Internal Technical Report, http://www.tek.com/VND/.