MANIPULATING TEMPORAL DEPENDENCIES IN COMPRESSED VIDEO DATA WITH APPLICATIONS TO COMPRESSED-DOMAIN PROCESSING OF MPEG VIDEO

Susie J. Wee

Hewlett-Packard Laboratories
Palo Alto, CA USA
swee@hpl.hp.com

ABSTRACT

The ability to manipulate the temporal dependencies in coded video data is important for a number of compressed-domain video processing tasks. This paper formulates the general problem and examines it in the context of MPEG. This is used to develop a method for performing frame conversions in MPEG coded video data. These frame conversions are used to develop compressed-domain video processing algorithms for performing temporal mode conversion, frame-by-frame reverse play, and frame-accurate splicing.

1. INTRODUCTION

Many modern video compression algorithms use predictive methods to exploit the similarities that commonly exist between neighboring frames of a video sequence. These algorithms achieve high degrees of compression, but in doing so, they create temporal dependencies in the coded data stream. When decoding a single video frame from the coded data stream, the temporal dependencies may require us to first decode one or more other video frames in the stream.

Consider now the problem of creating a new coded data stream from a given coded data stream, where the new stream contains only a subset of the video frames coded in the given stream. This occurs when performing a splicing operation on compressed video data. A difficulty occurs from the temporal dependencies in the coded data. In particular, some of the frames included in the subset may use a prediction that depends on frames that are not included in the subset; it is not possible to reconstruct these frames without their predictions. To address this problem, we process these frames so that their predictions only depend on other frames included in the splice. In more general terms, we manipulate the temporal dependencies of the frames affected by the video processing operation.

In this work, we present a method of manipulating temporal dependencies in predictively coded data. This is useful for performing a number of video processing tasks on compressed data streams. The approaches developed here are applicable to video compression algorithms that utilize predictive processing, including MPEG-1/2/4 and H.261/3. However, for ease of presentation and because of its use in the digital television industry, we focus on applications in MPEG-2 and MPEG-1, hereafter referred to as MPEG.

This paper begins by formulating the problem of manipulating temporal dependencies in compressed video data. We then show how this formulation can be used in the context of MPEG video. We present methods of performing frame conversions on MPEG video frames. Finally, we use these frame conversion methods to perform compressed-domain video processing and transcoding tasks on MPEG video streams. Tasks include temporal mode conversions, frame-by-frame reverse play, and frame-accurate splicing.

2. PROBLEM FORMULATION

Consider a video compression algorithm that uses predictive processing between video frames. The algorithm uses a set of prediction rules to encode a video sequence into a compressed representation. These rules allow many choices to be made during the encoding process. Different sets of choices lead to different compressed representations of the original video sequence. Each representation is compliant with the video compression algorithm; however, there may be instances in which one compressed representation is preferable over another.

A set of video frames is denoted by $F = \{F_1, F_2, ..., F_n\}$. The corresponding set of coded video frames is denoted by $\hat{F} = \{\hat{F}_1, \hat{F}_2, ..., \hat{F}_n\}$. The subscripts represent the order in which the frames are coded. The coding order may or may not be equivalent to the temporal order in which the frames are acquired or displayed.

Frame F_i is coded with a predicted and a residual component. For a given prediction mode, a prediction process P uses a set of anchor frames A_i which includes one or more previously coded frames, i.e. $A_i \subseteq \{\hat{F}_1, \hat{F}_2, ..., \hat{F}_{i-1}\}$. S_i describes the side information used in the prediction process.

If frame F_i is predicted with $P(A_i, S_i)$, then the resulting residual R_i is calculated by

$$R_i = F_i - P(A_i, S_i). \tag{1}$$

This residual is then coded into the data stream. The coded residual $\hat{R}_i$ leads to a reconstructed video frame $\hat{F}_i$, where

$$\hat{F}_i = P(A_i, S_i) + \hat{R}_i. \tag{2}$$

As stated earlier, the compression algorithm allows a number of prediction modes. In another prediction mode, a prediction process P' uses a different set of anchor frames A'_i and side information S'_i. The resulting prediction is different, and a new residual R'_i is calculated by

$$R'_i = F_i - P'(A'_i, S'_i). \tag{3}$$

Using the first prediction mode, frame F_i is described by two parts, its prediction and residual, where

$$F_i = P(A_i, S_i) + R_i. \quad (4)$$

Using the second prediction mode, frame F_i is described by a different prediction and residual, where

$$F_i = P'(A'_i, S'_i) + R'_i. \quad (5)$$

Neglecting coding distortions, the two representations describe the same video frame, but they distribute the data between the predicted and residual components differently. Both representations are compliant with the compression algorithm, but each has its own set of prediction dependencies determined by the anchor frames associated with its prediction mode. This idea, while quite simple, can lead to useful results in the area of compressed-domain processing. Specifically, the ability to manipulate the prediction dependencies in coded video data is the basis of many compressed-domain video processing tasks.

Compressed-Domain Approximation

Let us now consider the problem of converting a video frame coded with one set of prediction dependencies to one coded with a different set of prediction dependencies. Assume that we are given a coded video stream and that we do not have access to the original signal. The video frame F_i was originally coded with prediction $P(A_i, S_i)$ and residual $\hat{R}_i$. The problem now is to change the prediction dependence from anchor frames A_i to A'_i, and the side information from S_i to S'_i, i.e. the frame must now be coded with prediction $P'(A'_i, S'_i)$.

Equation 3 shows how to compute the new residual when given the original frame. If we assume that the reconstructed frame approximates the original frame, $\hat{F}_i \approx F_i$, then equation 2 can be used to approximate the new residual by

$$R'_i \approx \hat{R}_i + P(A_i, S_i) - P'(A'_i, S'_i). \quad (6)$$

Thus, in order to convert the prediction mode of a coded video frame, we simply need to change its residual so that it accommodates the change in prediction. We must also update the side information so that it reflects the new prediction mode. This update should convey the change in anchor frames and prediction process.

3. CDP OF MPEG VIDEO DATA

3.1. MPEG Prediction Rules

MPEG is a video compression standard based on motion-compensated prediction and transform coding [1]. Some relevant aspects of the standard are discussed briefly below.

MPEG allows video frames to be coded in one of three modes: intra-frame (I), forward predicted (P), and bidirectionally predicted (B). I frames are coded independently of other frames; P and B frames are coded predictively using block-based motion compensation (MC). A set of video frames can be coded in any sequence of I, P, and B frames. A typical prediction sequence for a series of frames is:

$$I_0\ B_1\ B_2\ P_3\ B_4\ B_5\ P_6\ B_7\ B_8\ I_9. \quad (7)$$

The subscripts represent the temporal order of the frames.

While the prediction sequence shown above is common, we stress that MPEG allows any prediction sequence to be used as long as the rules and system requirements are satisfied. This key fact allows coded frames to be converted between the I, P, and B prediction modes. Subsequently, we can design compressed-domain video processing algorithms by using appropriate sets of frame conversions.

The prediction rules for I, P, and B frames are:

- I frames do not use prediction.
- P frames can use forward prediction from the preceding I or P frame, A_{forw}. Forward motion vectors (FMVs) describe the prediction.
- B frames can use forward prediction from the preceding I or P frame, A_{forw}, and backward prediction from the following I or P frame, A_{back}. Forward and backward motion vectors (FMV and BMV) describe the forward and backward predictions.

Each frame is divided into 16×16 blocks of pixels called macroblocks (MBs). Macroblocks can be coded in the modes listed in the following table; the mode used for each macroblock is specified in the coded bitstream. The allowable coding modes for each macroblock depend on the frame type, and are shown below:

- I frames contain i MBs.
- P frames contain i or p MBs.
- B frames contain i, b_{forw}, b_{back}, or b, MBs.

In addition, we find it useful to define the following frame types and allowable macroblock coding modes:

- B_{intra} frames contain i MBs.
- B_{forw} frames contain i or b_{forw} MBs.
- B_{back} frames contain i or b_{back} MBs.

The significance of B_{forw} and B_{back} frames is that they only have forward or backward dependencies. It is important to note that a B_{forw} frame differs from a P frame in that a P frame is used as an anchor frame when coding other predicted frames, while a B_{forw} frame is not. A B_{intra} frame differs from an I frame for the same reason.

The anchor frames, side information, and prediction process for each macroblock coding mode are shown in the following table. In the table, the anchor frames and forward and backward motion vectors are defined as:

A_{forw} = preceding I or P frame
A_{back} = following I or P frame
FMV = forward motion vector
BMV = backward motion vector
MC = block-based MC prediction process

MB type	A	S	$P(A,S)$
i	-	-	-
p	A_{forw}	FMV	MC(A_{forw}, FMV)
b	A_{forw}, A_{back}	FMV, BMV	.5MC(A_{forw}, FMV)+ .5MC(A_{back}, BMV)
b_{forw}	A_{forw}	FMV	MC(A_{forw}, FMV)
b_{back}	A_{back}	BMV	MC(A_{back}, BMV)

3.2. MPEG Frame Conversions

In an MPEG video stream, the temporal dependencies can be manipulated with frame conversions. MPEG frame conversions are achieved by performing an appropriate set of macroblock conversions. For example, a P frame can contain i and p MBs, while an I frame can only contain i MBs. Thus, a $P \rightarrow I$ conversion requires converting the p MBs to i MBs. The original i MBs may be left unprocessed. A partial list of frame conversions and their corresponding set of required macroblock conversions are listed below:

Frame conversion	MB conversions
$P \rightarrow I$	$p \rightarrow i$
$B \rightarrow B_{\text{forw}}$	$b \rightarrow b_{\text{forw}}$ $b_{\text{back}} \rightarrow i$
$B \rightarrow B_{\text{back}}$	$b \rightarrow b_{\text{back}}$ $b_{\text{forw}} \rightarrow i$
$B \rightarrow B_{\text{intra}}$	$b \rightarrow i$ $b_{\text{forw}} \rightarrow i$ $b_{\text{back}} \rightarrow i$

Once we determine the desired conversion, we can use equation 6 to convert the MBs between various modes. A number of MB conversions are shown in the table below. In principle, we first reconstruct the original and new predictions. The new residual is then the sum of the original residual and the difference between the original and new predictions. The side information is updated to S' to indicate the new frame type and the new MB types. Since the MPEG prediction sequence is determined by the frame types, updating the frame types implicitly updates the anchor frames to A'. The result is recoded into the data stream.

Conversion	New Residual R'
$p \rightarrow i$	$R + \text{MC}(A_{\text{forw}}, \text{FMV})$
$b \rightarrow i$	$R + .5\text{MC}(A_{\text{back}}, \text{BMV}) + .5\text{MC}(A_{\text{forw}}, \text{FMV})$
$b \rightarrow b_{\text{forw}}$	$R + .5\text{MC}(A_{\text{back}}, \text{BMV}) - .5\text{MC}(A_{\text{forw}}, \text{FMV})$
$b \rightarrow b_{\text{back}}$	$R - .5\text{MC}(A_{\text{back}}, \text{BMV}) + .5\text{MC}(A_{\text{forw}}, \text{FMV})$
$b_{\text{forw}} \rightarrow i$	$R + \text{MC}(A_{\text{forw}}, \text{FMV})$
$b_{\text{back}} \rightarrow i$	$R + \text{MC}(A_{\text{back}}, \text{BMV})$

These frame conversions can be performed in the pixel or DCT domain. In the former, the frames are reconstructed to their pixel-domain representations, and conventional motion compensation is used to reconstruct the anchor frames and calculate the new residuals. In the latter, the anchor frames are stored in DCT-domain representations, and DCT-domain motion compensation techniques are used [2].

3.3. Discussion

Note that the anchor frames, the frames on which a prediction is based, are solely determined by the frame types of the surrounding frames and are not explicitly specified in the MPEG bitstream. This is a key concern when performing frame conversions on coded MPEG streams. Specifically, when performing a frame conversion on a single frame, one must consider the effects on its surrounding frames. As an example, consider the prediction sequence shown in 7. If we convert frame B_5 to P_5, then we must consider the consequences on other frames. Specifically, after performing this conversion, frame B_4 must use anchor frames P_3 and P_5, and frame P_6 must use anchor frame P_5. The rule of thumb is that conversions between I and P frames do not affect other frames, nor do conversions between B, B_{forw}, and B_{back} frames. However, conversions between any other modes may affect the predictions of one or more other frames, and therefore may require them to be processed appropriately.

The frame conversions shown in the table above remove temporal dependencies between the frames. For example, in the $B \rightarrow B_{\text{forw}}$ frame conversion, the original prediction was based on both a forward and a backward anchor frame, while the resulting prediction is only based on a forward anchor frame. In this conversion, the forward and backward motion vectors were given, and creating the resulting stream involves discarding the backward motion vectors and retaining the forward motion vectors.

Now consider the problem of adding temporal dependencies between video frames. This may require generating a new set of motion vectors between the current frame and its new anchor frame. For example, consider an $I \rightarrow P$ frame conversion. The original I frame did not use prediction, and therefore did not have an anchor frame or forward motion vectors. When converting this frame to a P frame, the MPEG prediction rules specify that the previous I or P frame will be the anchor frame for forward prediction. The task that remains is to determine an appropriate set of forward motion vectors.

A set of motion vectors can be generated by using conventional motion estimation techniques on the reconstructed video frames, however, this may be a computationally expensive task. Research is being performed on developing efficient DCT-domain motion estimation techniques. Another area of research lies in developing more efficient motion estimation methods that exploit the motion information given in the original compressed video stream. This problem, described as motion vector resampling, is currently under investigation by the author. A method of resampling motion vector fields for a compressed-domain reverse play operation is developed in [3]. While the topic of motion vector estimation is beyond the scope of this paper, it is a crucial part of achieving arbitrary frame conversions for various compressed-domain video processing operations.

4. COMPRESSED-DOMAIN PROCESSING APPLICATIONS

Video frames are typically coded with a prediction based on one or more previously coded frames. Thus, properly decoding one frame requires first decoding one or more other frames. This temporal dependence among frames severely complicates a number of video processing and transcoding tasks, such as temporal mode conversion, reverse play, and splicing. The frame conversion methods presented earlier are used to manipulate the temporal dependencies of coded data streams. This is the basis for developing efficient compressed-domain algorithms that perform video processing and transcoding tasks on MPEG-coded video streams.

4.1. MPEG System Issues

The previous section discussed the prediction rules of the MPEG video compression standard. In addition to issues of temporal dependencies, MPEG also addresses system-level issues such as buffer requirements, coding order, and bitstream syntax requirements. A complete compressed-domain video processing solution must address all of these issues to create fully compliant MPEG data streams.

Temporal dependency issues are resolved by manipulating the prediction dependencies with the frame conversion methods discussed earlier. Buffer constraints are satisfied by using rate control techniques such as requantization. Coding order rules are satisfied by reordering the new coded data. Bitstream syntax requirements are satisfied by updating the appropriate header information such as time stamps and picture and macroblock types.

The compressed-domain video processing and transcoding tasks require the following steps:

1. Determine and perform an appropriate set of frame conversions.
2. Perform rate control by requantizing the DCT coefficients.
3. Reorder the new coded data.
4. Update the relevant header information.

Steps 2, 3, and 4 are common to the following tasks and are beyond the scope of this paper. In the following sections we focus on the frame conversions that need to be performed for each compressed-domain operation.

4.2. Temporal Mode Conversion

The ability to transcode between various temporal modes adds a great deal of flexibility to video communication and processing applications. For example, it provides a method of achieving various tradeoffs in rate and robustness. An MPEG sequence consisting of all I frames, while least efficient from a compression viewpoint, is more robust to channel impairments in a video communication system. In addition, an I-frame MPEG video stream facilitates many video editing operations, e.g. DCT-domain image processing algorithms [4] can be applied to each frame of the sequence to achieve the same effect on the entire sequence.

Temporal mode conversions can be used to accommodate particular rate/robustness profiles or IPB prediction sequences. However, performing conversions between some sets of modes may require generating new sets of motion vectors. Motion estimation can be performed on fully reconstructed frames in the pixel domain, partially reconstructed frames in the DCT domain, or by using motion vector resampling techniques that use the motion vector and residual information in the original coded data stream.

4.3. Frame-by-Frame Reverse Play

The goal of reverse-play transcoding is to create a new MPEG stream that, when decoded, displays the video frames in the reverse order from the original MPEG stream. The problem is difficult because of the prediction dependencies in the coded data. Reverse-play transcoding can be achieved by performing an appropriate set of frame conversions. First, we convert P frames to I frames. Then, we exploit the symmetry of the B-frame prediction process and simply exchange their forward and backward motion vectors. We then appropriately reorder the data, perform rate control, and update the header information. A detailed description of the reverse-play algorithm is given in [5].

The resulting MPEG stream will be slightly larger than the original stream because of the extra data that results from the $P \rightarrow I$ frame conversion. If the increase in data rate is not acceptable, the P frames can be converted to P frames in the reverse order. The problem then becomes one of estimating the motion vectors of the reversed P frames. One approach is to perform full motion estimation to get the new forward motion vector field. Computational savings can be achieved with fast approximate motion estimation techniques that use the motion vector information contained in the original MPEG video stream [3].

4.4. Frame-Accurate Splicing

The goal of the splicing operation is to form a video data stream that contains the first N_{head} frames of one video sequence and the last N_{tail} frames of another video sequence. For uncoded video, the solution is a simple cut-and-paste operation. For MPEG-coded video the problem is difficult because of the prediction dependencies in the coded stream. Frame-accurate, compressed-domain splicing can be achieved by performing an appropriate set of frame conversions. These conversions must remove the prediction dependencies on dropped frames. The basic steps of the splicing operation are given below:

1. Process the head data stream. If the last frame is an I or P frame, there is no dependence on dropped frames. If it is a B frame, convert the last string of consecutive B frames to B_{forw} frames.
2. Process the tail data stream. If the first frame is an I frame, there is no dependence on dropped frames. If it is a P frame, convert it to an I frame. If it is a B frame, convert the first string of B frames to B_{back} frames and convert the first P frame to an I frame.
3. Match and merge the head and tail data. Reorder the data, perform the rate control, and update the header information.

A more detailed description is given in [6].

5. REFERENCES

[1] MPEG-2 International Standard, Video Recommendation ITU-T H.262, ISO/IEC 13818-2, January 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] S. Wee, "Reversing motion vector fields," in IEEE International Conference on Image Processing, (Chicago, IL), October 1998.

[4] S.-F. Chang and D. Messerschmitt, "Manipulation and compositing of MC-DCT compressed video," IEEE Journal on Selected Areas in Communications, vol. 13, January 1995.

[5] S. Wee and B. Vasudev, "Compressed-domain reverse play of MPEG video streams," in SPIE Voice, Video, and Data Communications Conference, (Boston, MA), Nov 1998.

[6] S. Wee and B. Vasudev, "Splicing MPEG video streams in the compressed domain," in IEEE Workshop on Multimedia Signal Processing, (Princeton, NJ), June 1997.