IEEE Workshop on Multimedia Signal Processing, Princeton, NJ, June 1997.

SPLICING MPEG VIDEO STREAMS IN THE COMPRESSED DOMAIN

Susie J. Wee and Bhaskaran Vasudev
Hewlett-Packard Laboratories
1501 Page Mill Road, MS 3U-4
Palo Alto, CA 94304
swee@hpl.hp.com

Abstract - An algorithm is proposed for efficiently splicing two MPEG-coded video streams. The algorithm only processes the portion of each video stream that contains frames affected by the splice and it operates directly on the DCT coefficients and motion vectors of those frames. The algorithm achieves good visual performance with very low computational complexity, and incorporates rate control to prevent buffer underflow and overflow. If additional processing power is available, it can be used to further improve the quality of the spliced stream.

INTRODUCTION

Splicing is a technique commonly used in video editing applications. When splicing uncompressed video, the solution is obvious: simply discard the unused frames and concatenate the remaining data. In other words, for uncompressed video, splicing is simply a cut-and-paste operation. The simplicity of this solution relies on the ordering and independence of the uncompressed video data.

Modern video compression algorithms such as MPEG use predictive methods to reduce the temporal redundancies inherent to typical video sequences. These methods achieve high degrees of compression, but in doing so, they create temporal dependencies in the coded data stream. These dependencies complicate a number of operations previously considered simple – splicing is one such operation.

In this work, we examine the problem of splicing two MPEG-coded video data streams. The only previous work we found published in the literature on this topic was as part of a video editing system by Meng and Chang [1]. In this work, we develop a compressed-domain splicing algorithm that provides a natural tradeoff between computational complexity and compression efficiency; this algorithm can be tailored to meet the requirements of a particular system. This paper begins by describing the problem of splicing MPEG-coded video streams. The proposed algorithm is then presented, and the frame conversion and rate control issues are discussed. The splicing algorithm was implemented in software and experimental results are given.

SPLICING ALGORITHM

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. A naive splicing solution is to completely decompress the two video streams, cut-and-paste the decompressed video frames in the pixel domain, and recompress the result. With this method, each frame in the spliced video must be completely *recoded*. This method has a number of disadvantages including high computational requirements, high memory requirements, and low performance due to recoding.

In the proposed algorithm, the computational requirements are reduced by only processing the frames affected by the splice. For example, in most instances the only frames that need to be processed or recoded are those in the GOPs affected by the head and tail cut points; at most, there will be one such GOP in the head data stream and one in the tail data stream. Furthermore, the data need not be decompressed into its pixel domain representation; improved performance is achieved by only *partially decompressing* the data stream into its motion vector and sparse DCT representation.

The splicing algorithm has three parts: form the head data stream which contains a series of frames from one sequence, form the tail data stream which contains a series of frames from another sequence, and match these data streams to form an MPEG-compliant data stream. The splice cut points of the head and tail streams can occur at any point in the IPB video sequence. One difficulty in splicing MPEG-coded video stems from the fact that video data needed to decode a frame in the spliced sequence may depend on data from frames not included in the spliced sequence.

In the proposed algorithm, the temporal dependence problem is solved by converting the picture types of the appropriate frames. The allowable conversions retain the ordering of the original coded data streams for frames included in the spliced sequence, thereby simplifying the computational requirements of the splicing algorithm. The algorithm results in an MPEG-compliant data stream with variable-sized GOPs, exploiting the fact that the GOP header does not specify the number of frames in the GOP or its structure, rather these are fully specified by the ordering and type of the picture data in the coded data stream. The algorithm uses rate control to ensure that the resulting spliced stream satisfies the specified buffer constraints. The steps of the proposed splicing algorithm are described below.

1. Form the head data stream The simplest case occurs when the cut for the head data occurs immediately after an I or P frame. When this occurs, all the relevant video data is contained in one contiguous portion of the data stream. The irrelevant portion of the data stream can simply be discarded, and the remaining relevant portion does not need to be processed. When the cut occurs immediately after a B frame, some extra processing is required because one or more B-frame predictions will be based on an anchor frame that is not included in the final spliced video sequence. In this case, the

leading portion of the data stream is extracted up to the last I or P frame included in the splice, then the remaining B frames are converted to P frames as described below in the *Frame Conversion* section.

2. Form the tail data stream. The simplest case occurs when the cut occurs immediately before an I frame. When this occurs, the video data preceding this frame may be discarded and the remaining portion does not need to be processed. When the cut occurs before a P frame, the P must be converted to an I frame, and the remaining data remains in tact. When the cut occurs before a B frame, extra processing is required because one of the anchor frames are not included in the spliced sequence. In this case, if the first non-B frame is a P frame, convert it to an I frame. Then, convert the first consecutive B frames to B_{back} mode as described below in the *Frame Conversion* section.

3. Match the head and tail data streams. The IPB structure and the buffer parameters of the head and tail data streams determine the complexity of the combining operation. When using the steps described above, this step requires first concatenating the two streams and then processing the data near the splice point to ensure that the buffer constraints are satisfied. This requires *matching* the buffer parameters of the pictures surrounding the splice point. In the simplest case, a simple requantization will suffice. However, in more difficult cases, a frame conversion will also be required to prevent decoder buffer underflow. This process is discussed further in the *Rate Control* section.

FRAME CONVERSION

The splicing algorithm may require converting frames between the I, P, and B prediction modes. Converting P or B frames to I frames is quite straightforward, however, conversion between any other set of prediction modes is much more complicated. Exact algorithms require performing motion estimation on the decompressed video – this process can dominate the computational requirements of the splicing algorithm. In this work, we use approximate algorithms that significantly reduce the number of computations required for this conversion. If additional computational power is available, then additional processing can be used to improve quality.

The first two steps of the proposed splicing algorithm may require B-to-P, P-to-I, and B-to-B_{back} frame conversions; and in the next section it will be shown that the third step may require I-to-P frame conversions. Each of these conversions is described below. In these conversions, the data is only *partially decoded* into its DCT and motion vector (DCT+MV domain) representation. This data is then converted into its new DCT+MV or DCT domain representation, and the result is recoded into the MPEG stream. The frame conversions will often require an inverse motion compensation operation. When this is needed, an efficient DCT-domain algorithm is used

to calculate the new DCT coefficients of the motion-compensated prediction directly from the intraframe DCT coefficients of the anchor frame [2].

P to I Frame Conversion In order to decode a P frame, the previous I and P frames in the GOP must first be decoded. Similarly, when converting a P frame to an I frame, the previous I and P frames in the GOP are first partially decoded into their DCT-domain representations. Then, for each P frame the DCT-domain inverse motion compensation algorithm is used to calculate the DCT coefficients of the motion compensated prediction, which is then added to the partially decoded residual DCT coefficients to form intraframe DCT coefficient representation. Finally, the intraframe DCT coefficients are recoded into the MPEG stream.

I to P Frame Conversion The difficulty of I to P frame conversion stems from the fact that no motion vectors are available for forward prediction. Because of the high computational cost of performing motion estimation, each macroblock of the frame is initially coded in intraframe mode or in forward prediction mode with a motion vector of (0,0). If additional computational power is available, a fast DCT-domain motion estimation algorithms is used to find better motion vectors to improve the motion compensated prediction of each block.

B to P Frame Conversion In P frames, each macroblock can be coded in intraframe mode or with forward prediction. In B frames, each macroblock can be coded in intraframe mode or with forward, backward, or bidirectional prediction. The B-to-P frame conversion requires each macroblock of the B frame to be converted to an allowable P-frame macroblock. Thus, if the B macroblock was coded with forward prediction or in intraframe mode, it can be left as is, but if the macroblock was coded with backward or bidirectional prediction, it must be converted to an intraframe or forward predicted macroblock. In our conversion, the bidirectional macroblocks are converted to forward predicted macroblocks using the given forward motion vector. The new prediction is calculated and the residual DCT coefficients are adjusted appropriately. If the macroblock used backward prediction, then it is coded either in intraframe mode or with forward prediction with a motion vector of (0,0). As in the I to P conversion, if additional computational power is available, a fast DCT-domain motion estimation algorithm is used to refine the motion vector.

B to $\mathbf{B_{back}}$ Frame Conversion The B_{back} frame is defined as one in which the macroblocks can be coded with backward prediction or in intraframe mode – this is essentially the dual of a P frame but for backward prediction. Thus, the B to B_{back} conversion is simply the dual of the B to P conversion in which the forward and backward instances are reversed.

RATE CONTROL

In MPEG, each picture is coded into a variable-length data segment. However, the frame rate of the displayed sequence is constant. Achieving constant bit rate (CBR) transmission of the MPEG stream requires buffers at both the encoder and the decoder. In CBR transmission, the decoder buffer is filled at a constant rate and the I, P, or B picture data is emptied at regular time intervals corresponding to the frame rate of the sequence. If a picture contains a large amount of data, the buffer empties by a large amount. Thus, a stream that contains many large frames in close succession may cause the buffer to underflow, i.e. in a CBR channel, the picture data may not be received in time to be displayed.

The MPEG syntax requires the buffer size to be specified in the sequence header, thus it is specified once at the beginning of the stream and can not be changed. MPEG also requires a vbv_delay parameter to be specified in each picture header; vbv_delay indicates the length of time the picture start code must be stored in the buffer before it is decoded [3].

In the proposed splicing algorithm, a number of frames are converted between various temporal modes. I frames typically require more data than P or B frames; thus, if many I frames are created, the resulting data stream may cause the buffer to underflow. The splicing algorithm must incorporate some form of rate control to prevent buffer underflow and overflow.

The problem becomes one of matching the buffer parameters of the head and tail data streams. More precisely, the head and tail frames near the splice point should be processed so that the vbv_delay parameter of the remaining tail frames retain their original values. Thus, if we assume that the two original streams satisfy the buffer constraints, then by properly matching the head and tail streams we ensure that the spliced stream also satisfies buffer constraints.

Meng and Chang solve the rate control problem by inserting synthetic fade-in pictures with low data rates [1]. They mention the possibility of applying a rate modification algorithm by using data partitioning, but do not develop this further.

In this algorithm, we achieve rate control with one of two methods. If the buffer occupancy of the pictures surrounding the splice point are similar, we requantize the DCT coefficients in the frames near the splice point to obtain a suitable match. If the resulting stream satisfies the buffer constraints and has acceptable video quality, then the splicing operation is complete. If the buffer occupancies are very different or if the requantized stream still causes the buffer to underflow, then a more drastic operation is needed to lower the bit rate. In this case, the I frames near the splice point in the head and tail streams are converted to P frames. Specifically, the last I frame in the head sequence and the second I frame in the tail sequence are converted to P frames. In addition, requantization is used to fine tune the buffer matching problem. Note that these frame conversions do not change the coding order of the picture data.

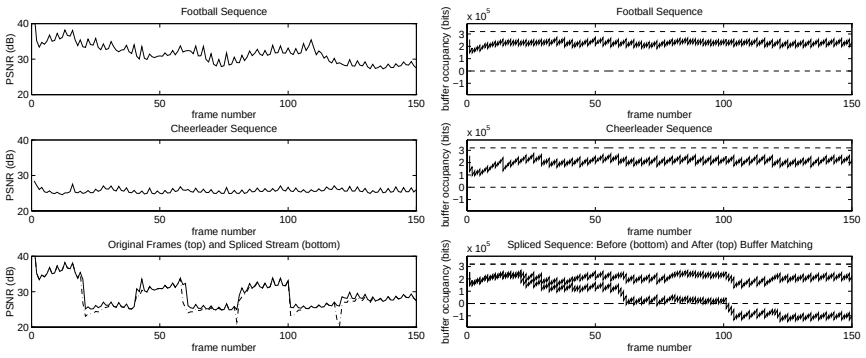

Figure 1: *PSNR and buffer occupancy of the original and spliced sequences.*

EXPERIMENTAL RESULTS

The proposed algorithm was used to splice the MPEG-coded football and cheerleader video streams. Both sequences had resolutions of 352 by 240 pixels at 30 fps and were coded with a target data rate of 1.152 Mbps. The PSNR and buffer occupancy of the streams are shown in the top of Figure 1. The streams have 15-frame GOPs with one I frame, 4 P frames, and 10 B frames. The spliced stream switches between the football and cheerleader sequences every twenty frames.

If the streams are spliced without addressing rate control, the buffer underflows as shown in the lower trace in the lower right plot in Figure 1. When applying rate control with requantization, the buffer underflow problem is eliminated. The resulting PSNR is shown in the lower left plot in the figure. The solid line represents the PSNR of the original coded frames, and the dotted line represents the PSNR of the spliced stream.

ACKNOWLEDGMENTS

The authors would like to thank Sam Liu for providing the MPEG parsing software that was used in these simulations.

References

[1] J. Meng and S.-F. Chang, "Buffer control techniques for compressed-domain video editing," in *Proceedings of IEEE International Symposium on Circuits and Systems*, (Atlanta, GA), May 1996.

[2] N. Merhav and B. Vasudev, "Fast inverse motion compensation algorithms for MPEG-2 and for partial DCT information," *HP Laboratories Technical Report*, vol. HPL-96-53, April 1996.

[3] *MPEG-2 International Standard,* Video Recommendation ITU-T H.262, ISO/IEC 13818-2, January 1995.