Stream Conversion to Support Interactive Video Playout

Ming-Syan Chen
National Taiwan University

Dilip D. Kandlur
IBM Thomas J. Watson Research Center

Interactive playout of MPEG-encoded video entails new ways of handling data. Transforming the standard MPEG stream to a local form at the player device enables efficient interactive playout even when available buffer space is constrained. A stream conversion scheme that encodes P frames as I frames after decompression and playout of each P frame eliminates extra memory needs, making P-I conversion a cost-effective solution.

Recent advances in computing, storage, and communication technologies have made many exciting multimedia applications possible. Compression techniques play a key role in processing digital multimedia data, particularly video data. Several factors drive the use of compressed video data. These include the prohibitively large storage required for uncompressed video data, relatively slow storage devices that are unable to retrieve uncompressed video data for real-time playout, and network bandwidth that does not allow real-time video transmission for uncompressed data. For example, a single uncompressed color frame with 620 × 560 pixels and 24 bits per pixel requires approximately one Mbyte of storage. At a full-motion playout rate of 30 frames per second, a 30-minute video would require more than 50 Gbytes of storage.

ISO MPEG[1] is the prevalent standard for compressed video with audio. MPEG's inter-frame compression techniques provide significant advantages in storage and transmission. To facilitate storage and retrieval, the MPEG standard defines a compressed stream whose rate is bounded. By reducing file size and thus memory needs for storage and transmission, MPEG makes possible the transmission of video with audio over networks.

New services, new computing needs

Interactive TV and video-on-demand (VOD) are two important services made possible by advances in video compression and network transmission technologies.[2] In a VOD system, multimedia streams are stored on a storage server (the video server) and played out to the user station on request. A significant amount of effort has gone into refining video server design.[3,4] A VOD server is expected not only to concurrently serve many clients (hundreds or more), but also to provide interactive features for video playout including pause/resume, backward play, and fast-forward and fast-backward play, which home viewers have come to expect from their current VCR systems.

However, recent studies indicate that to meet these requirements, the server would need a tremendous amount of computing power, storage, and communication bandwidth.[5,6] The inter-frame dependencies of MPEG make it prohibitively expensive to provide certain interactive features over the network.[5] Furthermore, such factors as skewed movie requests and peak-hour activities have made it very difficult, if not impossible, to have a cost-effective resource allocation (in terms of CPU, storage, and network bandwidth) in a VOD system. The feasibility of providing interactive movie viewing over networks (including backbone and cable networks) is unclear and requires technical answers to the issues raised above.

Video data distribution

To avoid the above drawbacks, in this article we consider an alternative solution for movie-on-demand service. This solution involves downloading the video data for storage in a player device located at the customer premises so that the customer can view the video without further intervention from the network. Such a player device can prove economically feasible. Using the MPEG compression technique, a 100-minute MPEG-1 movie will occupy 1.0 to 1.5 GBytes of storage. Currently, the price of a 1.6 GByte disk is under $400; rapid increases in recording densities will bring this cost down even further, making it more attractive to consumers.

With current disk bandwidths (such as on a SCSI disk), downloading a 100-minute MPEG-1

1070-986X/96/$5.00 © 1996 IEEE

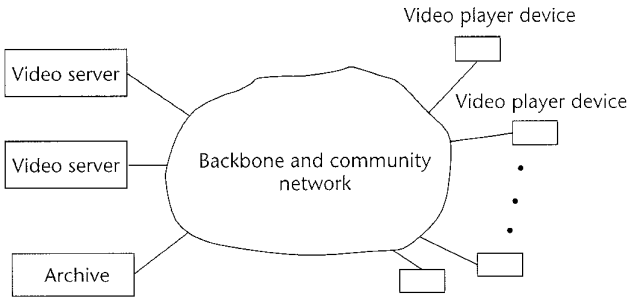


Figure 1. A schematic view of a movie-on-demand system. Compressed video data flows from servers through a network to the client stations, where it is stored for decompression and playout.

movie from the remote video server over the network to the disk of a client station will take only three to five minutes. This approximates the typical length of TV commercial breaks, an interval generally acceptable to end viewers. Since the video data now resides in the player's storage, viewers can enjoy all the interactive features for video viewing without incurring any server resource or network bandwidth problems. In addition, since downloading can be done prior to viewing, the effects of skewed movie requests and peak-hour activities can be minimized. Unlike real-time VOD, the downloading model permits marketing promotions such as discounts for slow business hours that do not require viewers to change their viewing time.

Figure 1 illustrates an environment for a movie-on-demand system that has a video player device at the customer premises. In this environment, video data is stored in the video server and transmitted to the player device on request. Note that under the downloading model, the VOD server is able to send videos out as fast as possible without explicit pacing. Also, since this is not real-time playout, the server has the flexibility of queuing up requests for downloading. These capabilities make downloading digital movies into a customer-based player device economical and desirable, and thus a competitive choice for real-time VOD.

Playback requirements

Given this downloading model, it is important to enhance the performance of such a player device by minimizing its system resource requirements. Even with this device, however, a deficiency exists for interactively playing MPEG movies, specifically for backward and fast-backward playout. Due to the inter-frame dependency of MPEG, backward playout requires a large amount of buffer memory to store many decompressed frames for decoding a frame. (It should be pointed out, however, that such a deficiency is not unique to the downloading model and in fact also exists for real-time VOD.) Since the video player, like most consumer products, is a very price-sensitive component, such a requirement for large memory buffers would be highly undesirable for product competitiveness.

To address this issue, we propose an efficient method that uses stream conversion to support interactive playout for MPEG videos and minimizes the buffer requirement in the player device. In essence, we seek to employ a compressed stream compatible with video data distribution standards and locally enhance it for special effects. The standard stream is defined to be highly compressed, so as to minimize the cost of distribution, such as storage and network transmission; the local stream, on the other hand, is optimized for effective playback.

An MPEG video stream consists of intra (I) frames, predictive (P) frames, and interpolated (B) frames. In this stream, I frames are coded such that they are independent of any other frames in the sequence. P frames, coded using motion estimation, are dependent on the preceding I or P frame. B frames, on the other hand, depend on two "anchor" frames: the preceding I/P frame and the following I/P frame. Since P and B frames use inter-frame compression, they are substantially smaller than I frames.

While the transformation of a standard compressed stream into a local stream can be realized in many ways, we use it to devise a scheme that encodes P frames as I frames after the decompression and playout of each P frame, thus efficiently supporting interactive playout for MPEG videos. We call this process of transforming a P frame to an I frame P-I conversion. Since P-I conversion is performed after a P frame is decompressed and played out, decoding requires no extra cost. More importantly, since no motion estimation or compensation is required for compressing a single frame into an I frame, this I frame encoding can be done very efficiently. Due to inter-frame dependencies, backward playout would normally require a large memory buffer. Using the technique of P-I conversion, backward playout requires no more memory buffer than normal playout.

It is possible to transform all P and B frames into I frames, that is, convert MPEG to JPEG,[7] after their playout. However, P-I conversion is preferable to MPEG-JPEG conversion in that it requires a much smaller amount of secondary storage in the player device and still enables backward playout operation with only the minimal amount of buffer required for MPEG forward playout. Since the backward operation is mostly employed in the form of fast-

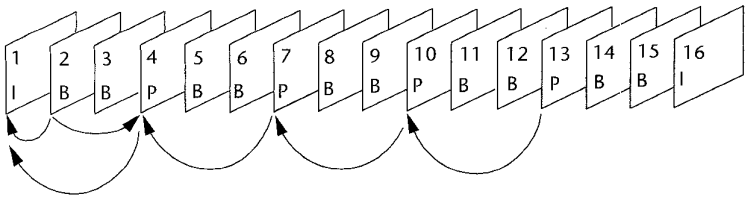


Figure 2. Inter-frame dependency in a sequence of MPEG frames.

Temporal Order:	I	B	B	P	B	B	P	B	B	P	B	B	P	B	B	I	...
Frame Number:	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	
Presentation/ Storage Order:	I	P	B	B	P	B	B	P	B	B	P	B	B	I	B	B	...
Frame Number:	1	4	2	3	7	5	6	10	8	9	13	11	12	16	14	15	

Figure 3. Temporal order (to the viewer) and presentation order (to the decoder) of MPEG frames.

backward playout, we conducted several video experiments, discussed later in this article, to evaluate various types of fast-backward playout and demonstrate the advantages of P-I conversion.

Most prior work on VOD focused mainly on resource management and task scheduling in the video server.[8,9] Little work discusses provisions required for the downloading model or deals with stream conversion to facilitate interactive playout in the client station. We propose that, being cost effective, easy to implement, and able to provide high visual quality interactive playout, P-I conversion is a viable approach to supporting interactive playout for MPEG video in a client station.

MPEG data organization

The structure of the MPEG stream imposes several constraints on video data storage and playout. Figure 2 shows the inter-frame dependencies in a sequence of MPEG frames, where the frames are numbered in temporal order. The arrows indicate the dependencies between frames. As mentioned, the inter-frame dependency implies that it is not possible to decode a P frame without the preceding I or P frame. Similarly, it is not possible to decode a B frame without the corresponding two anchor frames (that is, two P frames, or one I and one P frame). Storing these anchor frames requires some additional memory buffer.

Figure 3 shows the difference between the order in which compressed frames are presented to the decoder (presentation order) and the order in which decompressed frames are presented to the viewer (temporal order). For normal forward playout, it is necessary to keep exactly two decompressed frames in the memory buffer for decoding a frame that references these two frames. For the example in Figure 3, decompressed frames 1 (I frame) and 4 (P frame) are needed to decode frame 2 (B frame). On the other hand, when decoding frame 5 (B frame), we need decompressed frames 4 and 7 (two P frames) and no longer need frame 1. Since decompressed frames are the same size, we need buffer space for just two decompressed frames to do the decoding for normal playout.

While this presentation order reduces the buffer space required for forward playout, it does not address the problem of backward playout. Because frames are encoded using forward prediction, in order to move backwards in the stream to display a particular frame, it is necessary to decompress a large number of preceding frames on which this frame might depend. These decompressed frames are large, and they substantially increase the decoder's memory requirement. Moreover, the number of such buffers required increases linearly with the length of the chain of predicted frames. To resolve this deficiency of MPEG, we present in the following section a method of encoding P frames as I frames after the decompression and playout of each P frame. We will show how P-I conversion significantly reduces the buffer requirement of a player device and facilitates the interactive playout in the client station.

Compressed stream conversion

P-I conversion is a highly effective way to transform a standard compressed video stream to a local form at the player device. Similar conversions can be devised for other video compression algorithms based on motion compensation.

MPEG stream conversion

As explained earlier, although the presentation sequence of MPEG obviates the need to store compressed frames during forward playout, it does not address the problems of backward playout. For example, consider the case of a viewer who decides

Figure 4. A snapshot of the result of P-I conversion.

Original Frames Stored:	I	P	B	B	P	B	B	P	B	B	P	B	B	I	B	B	...
Frame Number:	1	4	2	3	7	5	6	10	8	9	13	11	12	16	14	15	
Frames Stored After P-I Conversion:	I	I	B	B	I	B	B	I	B	B	P	B	B	I	B	B	...
Frame Number:	1	4	2	3	7	5	6	10	8	9	13	11	12	16	14	15	

Figure 5. Temporal order (to the viewer) and presentation order (to the decoder) for backward playout.

Temporal Order:	I	B	B	I	B	B	I	B	B	I	B	B	I	B	B	I	...
Frame Number:	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	
Presentation Order:	I	I	B	B	I	B	B	I	B	B	I	B	B	I	B	B	...
Frame Number:	16	13	15	14	10	12	11	7	9	8	4	6	5	1	3	2	

to play backward when viewing frame 14, illustrated in Figure 3. At that moment, decompressed frames 13 and 16 are in the buffer, which means the viewer can then view frame 13. However, to decode frame 12, the decoder needs decompressed frames 10 and 13. To obtain decompressed frame 10, the decoder needs decompressed frame 7, which in turn requires decompressed frame 4 and frame 1. Thus, to decode any frame P during the backward playout, it is necessary to decode, in a reverse sequence, all the P frames until an I frame is reached. Note that this reverse chained-decoding is required for backward playout, but not for forward playout, since a P frame is encoded based on the previous I/P frame. The buffer space required for backward playout thus increases significantly; in this case, we need buffer space for five decompressed frames.

To facilitate backward playout, we propose transforming the standard MPEG stream into a local compressed form. Specifically, after a P frame is retrieved, decompressed, and played out, we encode this frame as an I frame and store it in secondary storage. As mentioned before, since this P-I conversion is performed after a P frame is decompressed and played out, there is no extra cost required for decoding. Also, since no motion estimation and compensation is required for compressing a single frame into an I frame, this I frame encoding can be done very efficiently.

Figure 4 shows a snapshot of the compressed frames stored in secondary storage when the normal playout reaches frame 14 (when we keep decompressed frames 13 and 16 in the buffer and frame 13 is yet to be converted into an I frame).

Using the proposed P-I conversion, the buffer space required for backward play will be the amount needed to store two decompressed frames—that is, the same as required for forward play. For example, consider again the viewer who decides to play backward when viewing frame 14 (with decompressed frames 13 and 16 in the buffer). Next comes frame 13, then frame 12 which is decoded based on frames 10 and 13. Note that with P-I conversion, frame 10 is now stored as an I frame in secondary storage and can be retrieved and decompressed by itself for decoding frame 12. The reverse chained-decoding required for the backward playout in the original MPEG stream is thus avoided. Figure 5 shows how the temporal order (to the viewer) differs from the presentation order (to the decoder) for backward playout after P-I conversion.

The additional storage required due to this P-I conversion is small. Assuming that the ratio of frame sizes for I, P, and B frames is 5:3:2, Table 1 shows the additional secondary storage required for P-I conversion and MPEG-JPEG conversion (in terms of the percentage over the size of the original MPEG stream) for various MPEG streams. The frame ratio in Table 1 indicates the mix of I, P, and B frames in the MPEG stream. For instance, the MPEG stream in Figure 2 has a frame ratio of 1:4:10. Consider this stream as an example. Since the ratio of frame sizes for I, P, and B frames is 5:3:2, the percentage of size increase by P-I conversion will be

$$\frac{5\times5+2\times10-(5\times1+3\times4+2\times10)}{5\times1+3\times4+2\times10}=21.6\%$$

On the other hand, the percentage of size increase by MPEG-JPEG conversion for this stream will be

$$\frac{5\times15-(5\times1+3\times4+2\times10)}{5\times1+3\times4+2\times10}=102.7\%$$

We can follow the same procedure to obtain the numbers in other columns of the table. These computations demonstrate that while P-I conversion requires the same amount of buffer for decoding, it requires significantly less secondary storage than MPEG-JPEG conversion. Note that a player device for MPEG-JPEG conversion also

requires the buffer for two decompressed frames to handle an incoming standard MPEG stream. Table 1 shows that the additional secondary storage required by P-I conversion is small and fairly acceptable in view of the resulting benefits and the inexpensive nature of the secondary storage.

Table 1. Comparing additional secondary storage requirements for P-I conversion and MPEG-JPEG conversion.

Frame ratio (I:P:B)	1:3:8	1:3:12	1:4:10	1:4:15
P-I conversion	20.0%	15.7%	21.6%	17.0%
MPEG-JPEG conversion	100.0%	110.5%	102.7%	112.7%

Implementation

The proposed P-I conversion method is easy to implement. To illustrate this, Figure 6 shows a block diagram of a player device designed to implement this transformation. This player contains an input device, temporary storage device, video/audio decoder, encoder, control logic, buffer memory, and display logic. The input device may be a network interface, a CD-ROM reader, or some similar device; it is used to read an incoming standard video stream. The temporary storage device, which is a read/write device, is used to store the transformed video stream.

The control logic implements the different operations on the device, while the conventional display logic is used to display images from the buffer. It is worth mentioning that depending on the applications, a downloading process could be performed in a pipelined manner so as to further reduce the local storage required. In this case, however, additional provision is needed to ensure the execution of interactive functions.

The execution flow for decoding during the normal playout is shown in Figure 7. The player reads consecutive video frames, decodes them, and displays them. As discussed earlier, the decoder operations, which are determined by the frame type, rely on two decompressed "anchor" frames. If the frame is an I frame, it is decompressed without depending on any other frame. The decompressed frame is

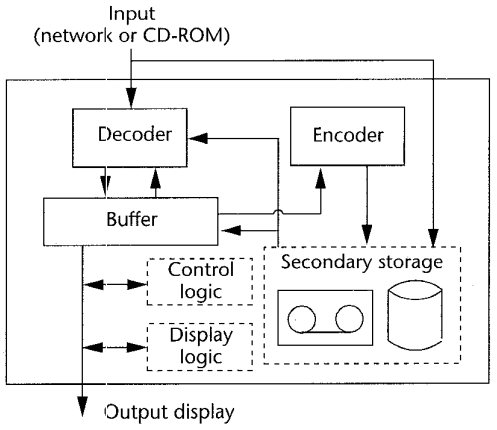


Figure 6. A player device with the features needed to transform a standard compressed stream into its local form.

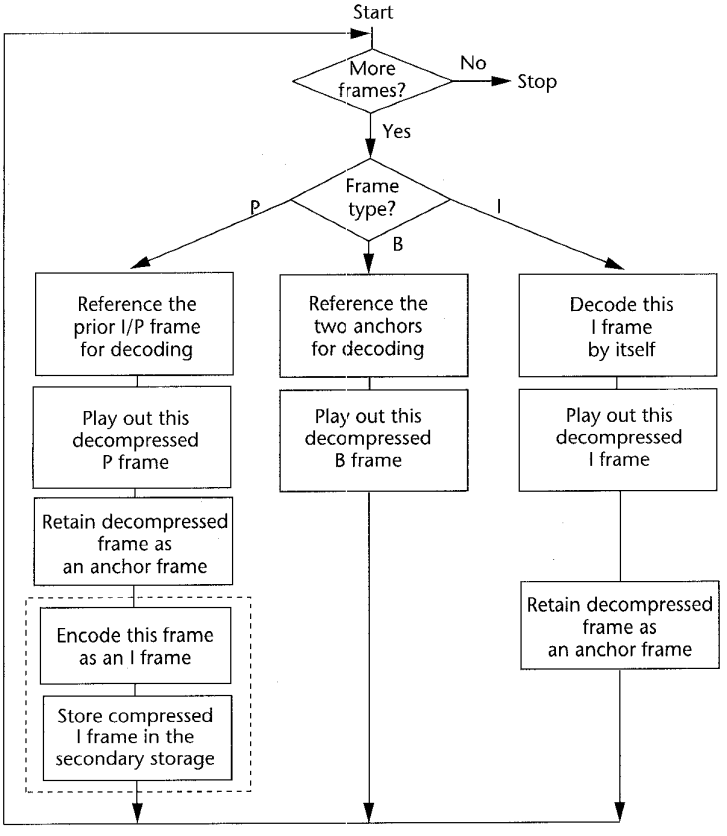


Figure 7. Execution flow for the decoder during forward playout.

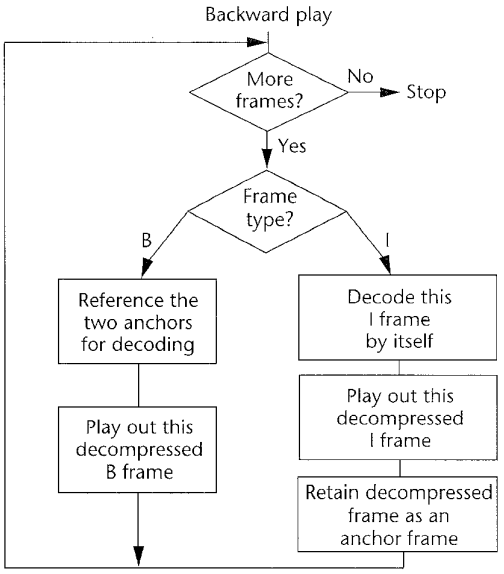


Figure 8. Execution flow for the decoder during backward playout.

retained in buffer memory as an anchor frame for successive frames. If the frame is a P frame, it is decompressed using the preceding anchor frame. This decompressed frame is also retained in buffer memory as an anchor frame.

In Figure 7, the dotted line indicates the extra operations that are required to convert P frames to I frames. The decompressed frame is now compressed as an I frame and stored in secondary storage. Since P-I conversion is performed after a P frame is decompressed and played out, it does not impose any additional cost or delay on the decoder. Also, since the encoding process does not require any CPU-intensive motion search/estimation, it can be performed easily in real time. It is not possible to perform in-place editing of P frames to I frames, since an I frame is generally larger than a P frame. Instead, we can copy a group of pictures (GOP), modify it, and then store it back. The original GOP can then be deleted.

Figure 8 shows the execution flow for decoding during the backward playout: The player reads successive video frames, decodes them, and displays them. The order of frame retrieval for backward playout is the reverse retrieval order for forward playout.

As in forward playout, frames are presented to the decoder in an order different from the temporal order. For example, in Figure 5, frame 13 is decoded before frame 15, since frame 13 is an anchor frame required for decoding frame 15. However, frame 15 is presented (displayed) before frame 13. Since P frames are replaced by I frames during forward play, the only frame types encountered during backward play are I and B frames. These frames are decoded and displayed in the same manner as that for forward playout.

In general, it is necessary to create offsets to I frames to access them efficiently during backward playout. However, such a need does not arise from the use of P-I conversion. In the original MPEG stream as well, creation of offsets to I frames is necessary for efficient frame access during backward playout. Although the GOP size changes in the modified stream, an MPEG decoder can still operate effectively. In addition to the stream level, an MPEG decoder can be kept informed of the picture pattern in the GOP level. Hence, depending upon the type of frame to be processed, an MPEG decoder can take appropriate actions for intra-frame or inter-frame decoding.

Experiments for fast-backward playout

Typically, the backward operation is used mainly in the form of fast-backward playout. (This is consistent with the fact that most commercial VCRs only provide the fast-backward play option.) In light of this, we would naturally like to know the impact of P-I conversion on the visual effect of fast-backward playout. For comparison, we assume that the player device has only the amount of buffer required for MPEG forward playout. The chained-decoding scenario described earlier for fast-backward playout of MPEG video requires a much larger buffer amount, and so is not included here for comparison.

In our experiment, we considered three types of fast-backward playout:

1. α-type. Without downloading, we use the segment skipping method[5] to implement fast-backward playout.

2. β-type. Without stream conversion, we only play I frames during the fast-backward playout. This method can be used with or without downloading.

3. γ-type. With P-I conversion, a fast-backward playout is performed using the backward playout technique described earlier.

Figure 9 illustrates these three types of fast-backward playout. Independent of their visual effects, note that while α and γ types of fast-backward can provide variable speeds, the β type is restricted to certain speeds. More specifically, the fast-backward speedup that β-type can provide has to be a multiple of the number of frames that an I frame is associated with (that is, 15, 30, 45, and so on, as in Figure 9b). Also, when β-type fast-backward is performed over the network, it has to be operated either at a higher cost (that is, higher data rate) or at a slower frame rate, since the size of an I frame is larger than the average size of a combination of I, P, and B frames.

To perform this visual experiment, we used the MMT (Multimedia Multiparty Teleconferencing) system, a prototype desktop collaboration system developed at the IBM Thomas J. Watson Research Center.[10] The current MMT hardware consists of an IBM PS/2 computer equipped with two custom-built adaptors. One adaptor (VAC) interfaces with analog video and audio components and performs capture (digitization) and playback functions. The other (MMT) is for compression, networking, processing, and decompression of video and audio.

Figure 10 shows one of the possible data paths through the MMT system. In this scenario, the video application extracts video and audio data from the MMT adaptor and transports it across the network. This video application has been modified to (1) store the video data to a file, and (2) play out video data from a file.

In the first instance, the timing or pacing for the recording process is provided naturally by the video source, and data is generated whenever a new video frame is captured and compressed. However, in the case of stored video playout, no suitable timing signal is available from the operating system to pace the playout process. We therefore chose to exploit the video source to provide pacing for the playout.

The playout application configures the MMT adaptor for video capture and playout. It then interprets the incoming video data to demarcate frames and paces the playout by ensuring that the number of video frames played out to the MMT adaptor closely corresponds to the number of video frames captured. We developed a filter program for processing the stored video to produce an output file with the appropriate segment size and speedup. This filter represents an off-line implementation of the fast-backward mechanism.

Using these tools, we have experimented with the three types of fast-backward playout for various video clips. The differences among these three types of fast-backward depend on the video content. For video clips with very little motion, such as those showing weather maps and news announcements, the visual effect of these three methods is similar.

For video clips containing objects moving in one direction, such as track actions, swimming, and long-distance touch-down runs in a football game, the differences among these fast-backward methods becomes prominent. We perceived some zig-zag motion for α-type, that is, objects appear to be moving back and forth. Also, we observed that watching β-type is similar to watching a slide projector operating at a high speed. In contrast, γ-type fast-backward is very smooth and the most visually acceptable, showing that a better visual quality results from P-I conversion.

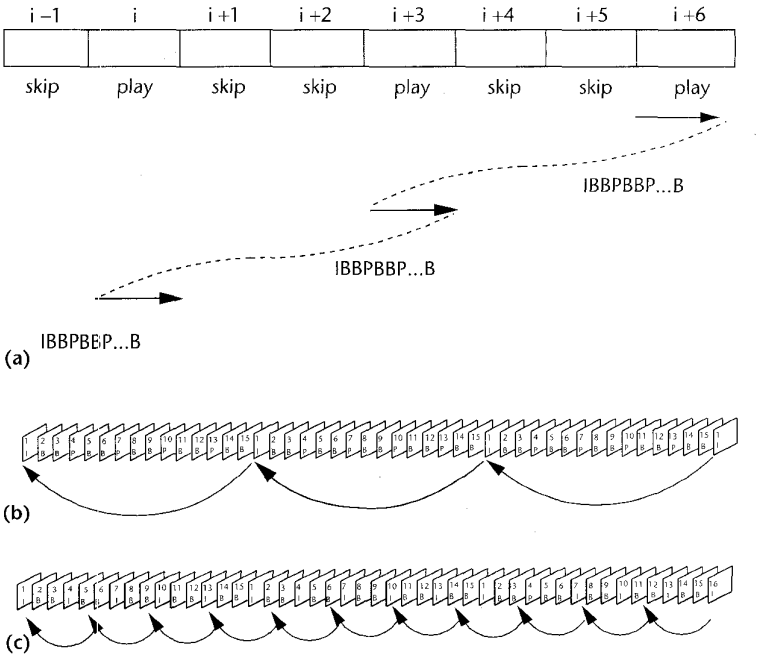


Figure 9. Three types of fast-backward playout evaluated: (a) α-type (fast-backward speedup: 3; starting from segment i+6), (b) β-type (fast-backward speedup: 15), and (c) γ-type (fast-backward speedup: 4).

Since the objective of our experiments is to determine the visual impact of different types of fast-backward, not the efficiency of the compression, using JPEG (in MMT) as the compression format will achieve the same visual effect. Also, the MMT provides us with facilities to capture various interesting scenes from television programs for evaluation. These visual experiments may also be performed using software MPEG decoders, such as one from the University of California at Berkeley.[11]

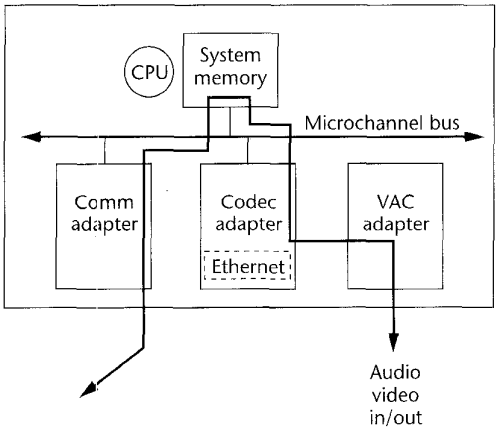


Figure 10. The MMT system used for the visual quality comparison experiment.

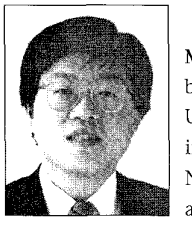
Conclusions

We suggest further exploration of the idea of using a consumer player device to provide interactive video functions for digital video streams. Our model involves downloading the video data directly to the storage of a player device located at the customer's residence. This lets the customer view the video thereafter without further intervention from the network.

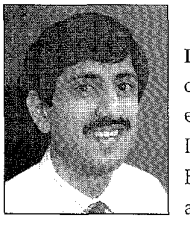
The P-I stream conversion method we proposed supports interactive playout, both forward and backward, of MPEG-encoded video streams and results in a substantial reduction of memory buffer requirement in the player device. Our experiments demonstrated that this method also gives acceptable visual quality. Based on its cost-effectiveness, ease of implementation, and ability to provide high-quality images, P-I conversion is a viable approach to supporting interactive MPEG video playout in a client station. **MM**

References

1. D. LeGall, "MPEG: A Video Compression Standard for Multimedia Applications," *Comm. of the ACM,* Vol. 34, No. 4, April 1991, pp. 46-58.
2. D. Deloddere, W. Verbiest, and H. Verhille, "Interactive Video on Demand," *IEEE Communications Mag.*, Vol. 32, No. 5, May 1994, pp. 82-88.
3. D.D. Kandlur, M.-S. Chen, and Z.-Y. Shae, "Design of a Multimedia Storage Server," *Proc. IS&T/SPIE Symp. on Electronic Imaging—Conf. on High Speed Networking and Multimedia Applications,* SPIE, Bellingham, Wash., 1994, pp. 164-178.
4. P.V. Rangan, H.M. Vin, and S. Ramanathan, "Designing a Multiuser Multimedia On-Demand Service," *IEEE Communications Mag.*, Vol. 30, No. 7, July 1992, pp. 56-65.
5. M.-S. Chen, D.D. Kandlur, and P.S. Yu, "Support for Fully Interactive Playout in a Disk-Array-Based Video Server," *Proc. ACM Multimedia,* ACM Press, New York, 1994, pp. 391-398.
6. J.K. Dey et al., "Providing VCR capabilities in large-scale video servers," *Proc. ACM Multimedia 94,* ACM Press, New York, 1994, pp 25-32.
7. G.K. Wallace, "The JPEG Still Picture Compression Standard," *Comm. of the ACM,* Vol. 34, No. 4, Apr. 1991, pp. 30-44.
8. B. Ozden et al., "A Low-Cost Storage Server for Movie on Demand Databases," *Proc. 20th Int'l Conf. on Very Large Databases,* VLDB, Santiago, Chile, 1994, pp. 594-605.
9. H.M. Vin and P.V. Rangan, "Designing a Multi-User HDTV Storage Server," *IEEE J. Selected Areas in Communication,* Vol. 11, No. 1, Jan. 1993, pp. 15-16.
10. M.-S. Chen et al., "Multimedia Desktop Collaboration System," *Proc. IEEE Globecom 92,* IEEE Press, Piscataway, NJ, 1992, pp. 739-746.
11. L. A. Rowe et al., "MPEG Video in Software: Representation, Transmission, and Playback," *Proc. IS&T/SPIE Symposium on High-Speed Networking and Multimedia Computing,* SPIE, Bellingham, Wash., 1994, pp. 134-144.



Ming-Syan Chen is a faculty member at the National Taiwan University. He received a BS degree in electrical engineering from National Taiwan University and MS and PhD degrees in computer, information, and control engineering from the University of Michigan. His research interests include multimedia technologies, database systems, and combinatorial theory. He was a research staff member at IBM's T.J. Watson Research Center from 1988 to 1996, where he worked on multimedia systems and data mining. IBM awarded him an Outstanding Innovation Award in 1994; he also has received awards for his inventions in the areas of interactive video playout, video server design, data mining, and multiprocessor networks. Chen is a senior member of the IEEE and a member of the ACM.



Dilip D. Kandlur received a BTech degree in computer science and engineering from the Indian Institute of Technology, Bombay. He received MSE and PhD degrees, also in computer science and engineering, from the University of Michigan, where he was awarded an IBM Graduate Fellowship. At IBM's T.J. Watson Research Center, his research has focused on multimedia systems, specifically video/audio support for desktop collaboration, multimedia networking, and multimedia storage management. He currently leads the Open Systems Networking group, whose focus is on network, transport, and higher layer protocols and application enablers for high-speed networks.

Contact Chen at the Electrical Engineering Department, National Taiwan University, Taipei, Taiwan ROC, e-mail mschen@cc.ee.ntu.edu.tw.