CIRCUITS SYSTEMS SIGNAL PROCESSING
VOL. 20, NO. 3, 2001, PP. 361–373

ONLINE RATE CONTROL FOR VIDEO STREAMS*

Sassan Pejhan,[1] *Tihao Chiang,*[2] *and Ya-Qin Zhang*[3]

Abstract. A mechanism for varying the frame rate of pre-encoded video clips online is described. The mechanism relies on two different encoders. An offline encoder creates a high-quality bit stream encoded at 30 fps, as well as separate files containing motion vectors for the same clip at lower frame rates. An online encoder decodes the bit stream (if necessary) and re-encodes it at lower frame rates in real time using the precomputed, stored motion information. Online frame rate control, used in conjunction with dynamic bit rate control, allows clients to solve the rate mismatch between the bandwidth available to them and the bit rate of the pre-encoded bit stream. Clients can therefore choose the amount of trade-off between temporal and spatial quality while resolving the rate mismatch problem. Moreover, online frame rate control provides a natural means for implementing a VCR-like fast forward control for video streaming applications, without increasing bandwidth consumption.

Key words: MPEG, rate control, video streaming, bit allocation, video server.

1. Introduction

The growth of the Internet and the proliferation of Internet browsers has enabled millions of users to access all kinds of information. Most of the information is currently in the form of text or still images and graphics. This type of information can be quickly downloaded and displayed with the click of a mouse button. Accessing and playing back audio and video clips, however, is not as straightforward.

There are two ways to access video clips across the Internet. The first is to

* Received February 26, 2000; revised October 1, 2000. An earlier version of this paper appeared in Proceedings of ACM Multimedia, 1999, pp. 141–144.

[1] Sorceron 459 W. 15th St. Suite 6E, New York, New York 10011, USA.
E-mail: spejhan@sorceron.com

[2] Department of Electronics Engineering, Nation Chiao Tung University, 1001 Ta-Hsueh Road, Hsinchu, Taiwan 30050, People's Republic of China. E-mail: tchiang@cc.nctu.edu.tw

[3] Microsoft Research, 5F, Beijing Sigma Center, No. 49, Zhichun Road, Haidian District, Beijing, 100080 People's Republic of China. E-mail: yzhang@microsoft.com

download the compressed bit stream in its entirety and then decode and display it. This method has a number of drawbacks if the clip is more than a few seconds long. One is that the user will have to wait a long time for the clip to be downloaded. Another is that the user needs to have enough free storage space to accommodate the entire video clip.

The second approach is referred to as *video streaming*; each frame of a video clip is immediately decoded and displayed upon reception. However, video streaming poses a new set of challenges. The original video sequence is compressed at a certain bit rate. However, users may need to download it at a different rate, depending on the bandwidth available to them. Reducing the delivery rate of the stream would result in a "slow motion" display at the receiver. Implementing VCR-type controls such as fast forward also becomes more complicated when streaming multimedia clips.

A common and widely investigated solution for the rate mismatch problem has been to partially decode the stream at the server and then re-encode it using different quantizer values, resulting in a different bit rate [2], [4]. This technique has proven to be quite powerful and accurate for a reasonable range of bit rates. It effectively provides a knob with which the user can vary the bit rate of the video clip. But this knob will only go so far before it causes a significant degradation in the spatial quality of the video clip.

To provide greater control (to the user) over the incoming video stream, a second knob, which would vary the frame rate of the sequence, is proposed in Section 2. By reducing the frame rate of the clip, the bandwidth consumption will be reduced without further loss in the spatial quality (temporal quality will be reduced instead). Moreover, this extra knob can be used for implementing a fast forward (FF) control, as described in Section 3. Section 4 describes the software implementation, the test bed for the experiments, and the performance results obtained. It also points out the limitations of the proposed scheme. Our conclusions are presented in Section 5.

2. Online frame rate control mechanism

Although one can partially decode and re-encode a video clip (using a different quantizer) in real time, decoding and re-encoding a stream at a different frame rate in real time is not as easy, unless the server is equipped with expensive special-purpose hardware. With certain encoding schemes, such as Motion JPEG [11] (in which each frame is encoded independently), this is not a problem: frames can simply be skipped in Motion JPEG.

However, such schemes, are much less efficient (in terms of compression) than schemes such as MPEG [7] and H.263 [10], which employ motion estimation techniques. Motion estimation, though, is an extremely time-consuming process, particularly if a full motion search algorithm is employed. With these schemes the sender cannot arbitrarily drop frames in the compressed domain because decoding of subsequent frames at the receiver depends on them. Thus, if the sender is to re-encode the video stream at a different frame rate, it would have to recompute the motion vectors. Performing this in real time, without special-purpose hardware, is not likely except for very low-resolution images.

One solution is to encode and store the sequence at different frame rates (say 30, 15, 10, and 6 fps). However, this will take up a lot of space on the server (see Section 4 for a quantitative analysis). The solution that is proposed here is to store only the motion vectors for the lower frame rates. This way all motion estimation is done offline and a priori. When re-encoding, motion vectors can be read from the motion files instead of being computed. This enables the stream to be re-encoded (at a different frame rate) much faster. The motion files are much smaller than the corresponding compressed streams, as shown in Section 4. Although this scheme is equally applicable to any video coding scheme using motion compensation, the implementation details that will be described here are specific to the H.263 standard.

2.1. Overview of H.263 coding scheme

The H.263 standard, specified in [10], is very similar to MPEG-1 and MPEG-2, although it is primarily geared towards very low-bit-rate coding. There are two types of frames in an H.263 stream: intracoded (I) frames and predicted (P) frames. (The use of a third type, PB-frames, is optional. PB-frames contain one predicted frame and one bidirectionally predicted frame. We will ignore these for the purpose of this paper.) Every single frame (regardless of the type) begins with a 17-bit picture start code (PSC). The I-frames are encoded very much like JPEG still images. A discrete cosine transform (DCT) operation is applied to 8×8 blocks of the frame. The DCT coefficients are then quantized and entropy encoded.

The P-frames are predicted from a preceding I- or P-frame. Using motion estimation techniques, each 16×16 macroblock (MB) in a P-frame is matched to the closest MB of the frame from which it is to be predicted. The difference between the two MBs is then computed. If the closest match for the MB to be coded happens to be the corresponding MB in the reference frame (i.e., the one for which the motion is zero), and if the difference between the two MBs is below a certain threshold, then the MB is not coded at all. Instead a single bit, which is called the COD field and set to 1, is inserted in the bit stream. When the decoder encounters a $COD = 1$ bit, it will simply repeat the MB of the preceding I- or P-frame.

If the closest match is some other MB (i.e., nonzero motion vector), then the COD bit is set to 0, and the motion vector is encoded and inserted into the bit stream. By default, one motion vector is encoded for one MB, which corresponds to four 8×8 blocks. There is an option (referred to as advanced prediction or AP), that allows for one motion vector per 8×8 block (i.e., four vectors per MB). The difference between the two MBs is then encoded in a similar fashion to blocks in the I-frames. Alternatively, if the difference (defined as the sum of absolute differences, or SAD) between the two MBs is larger than a threshold, then that particular MB will be encoded in INTRA mode (i.e., similar to MBs in I-frames) even though the overall frame itself is still a P-frame. In this latter case, no motion information is encoded.

2.2. *Details of proposed scheme*

As mentioned before, the scheme proposed in this paper is to store a high-quality, 30 fps compressed bit stream, but in addition, store only the motion vectors for the lower frame rates in separate motion files. Of course, the "motion files" need to contain more information than just the encoded motion vectors. The motion files contain the 17-bit start of picture code that is present for each H.263 encoded frame. This allows the video server to dynamically switch frame rates in the middle of a transmission. Also, since a given MB in H.263 can be coded in various forms (which will lead to a variable number of motion vectors for that MB), some information regarding it needs to be added to the motion vector file. This can be done in two different ways.

The first way is to add a single bit per MB indicating the presence or not of motion vectors. A "1" indicates that there is one motion vector for this MB. We have assumed that the AP mode of H.263 is not used (in AP mode there could be four motion vectors for each MB, and this would increase the size of the motion files). Similarly, we assume that PB-frame mode is not used. A "0" indicates that there are no motion vectors associated with the MB, which would be the case if the MB is not encoded ($COD = 1$) or if it is encoded in INTRA mode. To further distinguish between the $COD = 1$ case and the INTRA mode the encoder would have to perform SAD calculations for the zero vector and the INTRA case to decide whether the MB is to be encoded (INTRA) or not at all. However, these SAD calculations are time consuming, and will affect the performance of the online encoder.

An alternative solution is to insert a variable-length code (one or two bits) in the motion file to indicate both the COD and the MB type. A 0, for instance, would indicate $COD = 1$, a 10 would indicate $mode = INTRA$, and an 11 would indicate $mode = INTER$ and the presence of a motion vector. This results in a faster encoder, but at the price of a larger motion file. The format of the motion file is shown in Figure 1. Following the PSC (one for each frame), there is a 1 or 2 bit COD/MB type code per MB. If motion vectors are present, they are inserted following the COD/MB code. Otherwise, we move on to the COD/MB for the next MB. For the next frame, we again start with a PSC, and the pattern repeats itself.

In a video streaming application, the user can either explicitly specify the desired frame rate or provide the desired target bit rate to the sender and have the

17-Bit PSC	1/2 bit COD & MB	Motion Vector	. . .

Figure 1. Motion file format.

latter find an optimum between the frame rate and the quantization parameters. The sender would then decode the precompressed stream (encoded at 30 fps) frame by frame. For each frame, it will determine if the frame is to be encoded or not. For frames that need to be encoded and transmitted, it will open the motion file corresponding to the desired frame rate and encode the frame rates accordingly.

Because the precompressed stream is encoded at very high quality, it will be visually lossless. There will therefore be little difference between using that stream for re-encoding at lower frame rates and using the original video sequence. In other words, the quality of the lower rate stream will not suffer noticeably from the fact that it is based on a precompressd, as opposed to an original, stream.

One of the advantages of the proposed scheme is that because motion estimation is done off-line, a full search algorithm and a large search area can be used to obtain optimum compression efficiency. The encoders do not need to trade off quality for speed by using fast motion estimation algorithms.

3. Fast forward control

With the increased interest in playing back video clips on computers, be it from local storage or streaming clips over the network, there is a natural desire to have the same type of controls as one has with regular (analog) VCRs (such as play, stop/pause, fast forward, and rewind). Such controls can easily be implemented for raw video: the size of the frames is fixed, the position of each frame in the bit stream is known, and frames can be accessed and displayed independently from one another. For compressed video, implementing some of these controls can become challenging. Compressed frames have a variable size, and their position in the bit stream may not be readily available. Moreover, if predictive coding is used (such as motion compensation) a given frame may not be decodable independently of other frames in the sequence.

Digital video enables several new features that may supersede some of the traditional VCR controls. Automated video summarization techniques, for instance, provide a more practical method (than fast forward) for searching through a video sequence, by providing indexes to key frames [5]. Fast forward can still be useful for searching through long shots, however.

There are two ways to produce a fast forward effect. The brute-force method is to decode and display a video clip faster than its natural rate. If a clip is encoded at 30 fps, but the decoder runs at, say, 60 fps, the user will see the clip at twice the natural speed. The advantage of this method is that the decoder operates as before and on the same bit stream as that used in regular "play" mode. The disadvantages are twofold: to run the clip at rates significantly higher than the standard 30 fps, a powerful processor would be required, particularly for higher-resolution images. Moreover, in the case of streaming video, the bandwidth consumption would be increased. Running a 50 kbps clip at four times its natural frame rate would require 200 kbps of bandwidth. A variation on this theme has been proposed in [3] for the case of large-scale servers. A portion of the bandwidth is reserved for fast forward and fast backward operations. Fast forward and backward capabilities are provided with some probability. If the additional bandwidth required is not available, the service is either delayed or provided at a lower resolution.

The other technique is to skip frames. Skipping frames, however, becomes problematic with interframe coding techniques as discussed in Section 2. An easy solution is to just transmit and display the I- (intracoded) frames. However, because of their large size, these frames are few and far between (typically one or two out of 30 frames). The authors in [8] propose storing multiple versions of a video clip: one at 30 fps and others at lower frame rates. However, as pointed out in Section 2, this would significantly increase storage space. A 2-phase service model is proposed in [9] where nonadjacent portions of the video are first downloaded (and can provide a fast forward effect) during an initialization phase. The missing fragments are then downloaded during the second pass to provide regular "play" mode. The technique provides a middle ground between streaming and downloading. For longer sequences, the download time could be long, and the local storage requirements for the first phase will become significant. In [6] and [1] the authors divide the video clip into independently decodable segments, usually corresponding to one or more groups of pictures (GoPs). A GoP consists of one I-frame at the beginning and multiple P- and B-frames. No information outside the GoP is required to decode any of the frames in the GoP. Fast forward is then implemented by sampling the segments: 3$\times$ fast forward, for instance, is achieved by sending every third segment. Although, on the average, only one-third of the frames are transmitted and displayed, this scheme results in a nonuniform fast forward effect. Because there are few I-frames (as explained before), a GoP can be 10 or more frames long. In the 3$\times$ example above, frames 1 through 10 will all be transmitted and displayed (assuming a GoP size of 10 frames), whereas frames 11 through 30 will all be dropped, instead of transmitting and displaying every third frame (frame 1, frame 4, frame 7, etc.).

The online frame-rate-control scheme described above lends itself naturally to the frame-skipping technique for implementing fast forward control. If the server re-encodes a 30 fps clip at, say, 15 fps (by skipping every other frame), but the decoder decodes and displays it at 30 fps, the user will see the clip at twice its natural speed. The bandwidth requirements will be slightly more than that needed for regular "play" mode (the 15 fps clip will be slightly more than half the size of the 30 fps clip, but it needs to be sent twice as fast), and the decoder will still be operating at 30 fps. Similarly, if the user wants to see the clip at three times the natural speed, the server can re-encode the clip at 10 fps, and so forth.

4. Implementation and performance results

4.1. Codec software

The H.263 software codec that we used is the earlier H.263 V3.0 codec, not the H.263+ version which is developed by Telenor and currently maintained by the University of British Columbia.

Two modified versions of the encoder were created. One was the offline encoder which was used to create a high-quality coded bit stream at 30 fps. It was also used to produce the motion files for 15 fps, 10 fps, and 6 fps streams using a full-search algorithm to achieve the best results. A variable-length code representing the COD and MB type was used for the motion files (see the discussion in Section 2).

The other encoder was the online version, which was created by integrating the decoder with the encoder. The 30-fps-encoded bit stream could thus be decoded and re-encoded at lower frame rates (using the motion files). Also, the encoder was modified so as to transmit the encoded bit stream over the network instead of saving to disk. In addition, a bit-rate-control mechanism developed in-house [2] was integrated with the online encoder. A constant bit-rate stream could optionally be produced at a target specified by the user. The original Telenor codec includes two different rate-control mechanisms, one of them offline. A number of other minor optimizations were also made to improve performance. Overall, the complexity of the online encoder was vastly reduced by removing the motion estimation functions. It was slightly increased because an additional decoder module had to be added. However, the decoder, is less complex than the encoder. Furthermore, the decoder module integrated with the encoder was stripped of the image dithering and display modules (including YUV to RGB conversions), as well as the logic required to display frames at their correct rate. According to our profiler, these components consumed 50% of the original decoder's time.

On the decoder (client) side, the only modification made was to read the bit stream from the network instead of a local file.

4.2. Test bed

The tests had two main goals. One was to quantify the savings in terms of storage space of the proposed scheme as compared to storing several bit streams coded at different frame rates. The other was to compare the performance of the online encoder with that of the original H.263 encoder, and also to see if the former could be used for real-time streaming applications. The test platform was a 300-MHz Sun UltraSparc-2 workstation for the server (encoder) and a slower (200-MHz) version for the client. The server and client were linked over a moderately loaded internal local area network (LAN) (100 Mbits/s Ethernet), and no errors or packet losses were deliberately introduced. No other applications were being run on the server during the tests (except for the regular operating system-related daemons).

Three video sequences were used for our tests: Akiyo, News, and Coastguard. These are three of the sequences used in the MPEG-4 test suite. These are all 10-second, 300-frame sequences. Akiyo is a typical talking-head sequence (fixed camera and background, single talking person with small movement, no scene cuts, panning, or zooming) with minimal motion. The News sequence includes two news anchors (talking heads) with moderate motion and a high-motion ballet scene as background which also has scene cuts. The camera is fixed. The Coastguard sequence includes multiple objects (two boats) with high motion. The camera initially tracks one object and then the other. As such, the background is also changing. This sequence has very high motion content. Two versions of each sequence, at QCIF (176×144) and CIF (352×288) resolutions, were used for the tests.

4.3. Performance results

In all of our tests, the encoders were run without any of the H.263 options being turned on. Also, the same quantization parameters were used throughout.

4.3.1. Storage space

Table 1 shows the file size comparison between compressed bit streams at 30, 15, 10, and 6 fps and the corresponding motion files. As can be seen, the motion files are between 14 and 28%, 8 and 15%, and 8 and 11% of the size of their corresponding bit streams for the Akiyo, News, and Coastguard sequences, respectively.

Table 2 shows the reduction in required storage space for the proposed scheme (i.e., storing the 30 fps bit stream and only the motion files for lower frame rates) as compared to storing all four compressed bit streams. The storage space is reduced by more than 50% in general.

Note that, for all six sequences, the savings in storage increases as the frame rate drops. Moreover, the savings in storage is *higher* the greater the motion content of the sequence is. As the temporal difference between consecutive frames increases, more and more MBs would need to be encoded, because they are more likely to differ significantly from the closest-match MB in the previous frame. Motion vectors are also more likely to be nonzero and larger in general. However, overall, the ratio of the coded motion vectors to the rest of the bit stream falls with decreasing frame rates. The same explanation holds for sequences with higher motion content: there is more motion information, but more MBs need to be encoded, too.

Table 1. File size comparisons

File	Bitstream Size (bytes)	Motion file Size (bytes)	Ratio (%)
Akiyo QCIF 30 fps	28,375	6,509	22.9
Akiyo QCIF 15 fps	19,088	3,590	18.8
Akiyo QCIF 10 fps	15,575	2,567	16.5
Akiyo QCIF 6 fps	12,419	1,707	13.8
Akiyo CIF 30 fps	91,627	25,197	27.5
Akiyo CIF 15 fps	59,793	14,282	23.9
Akiyo CIF 10 fps	48,767	10,364	21.3
Akiyo CIF 6 fps	38,170	7,009	18.4
News QCIF 30 fps	78,503	9,155	11.7
News QCIF 15 fps	53,975	5,469	10.1
News QCIF 10 fps	44,092	4,023	9.1
News QCIF 6 fps	33,853	2,705	8.0
News CIF 30 fps	226,320	35,993	15.9
News CIF 15 fps	154,180	21,866	14.2
News CIF 10 fps	126,377	16,291	12.9
News CIF 6 fps	97,096	10,934	11.3
Coastguard QCIF 30 fps	184,078	19,462	10.6
Coastguard QCIF 15 fps	117,151	11,570	9.9
Coastguard QCIF 10 fps	96,099	8,752	9.1
Coastguard QCIF 6 fps	79,766	6,273	7.9
Coastguard CIF 30 fps	840,910	90,158	10.7
Coastguard CIF 15 fps	537,076	57,399	10.7
Coastguard CIF 10 fps	477,733	45,127	9.5
Coastguard CIF 6 fps	381,319	28,915	7.6

Table 2. Savings in storage for proposed scheme

File	4 Bit Streams	1 Bit Stream +3 Motion Files	Ratio (%)
Akiyo QCIF	75,457	36,239	48.0
Akiyo CIF	238,357	123,282	51.7
News QCIF	210,423	90,700	43.1
News CIF	603,496	275,411	45.6
Coastguard QCIF	477,094	210,673	44.2
Coastguard CIF	2,237,038	972,351	43.5

Note also that, in practice, the savings in storage would be even more than that shown in Table 2. The reason for this is that in our tests, all sequences were encoded such that the first frame was an I-frame and all the rest were P-frames. The frequency of I-frames is not specified by the standard and is up to the implementors. This achieved maximum compression, and it is how the H.263 original software was implemented. However, in practice, I-frames need to be inserted regularly (particularly if the streams are to be transmitted over noisy and/or congested lines) to prevent error propagation. I-frames are larger (several times on the average) than P-frames, so the compressed file sizes would be larger than those reported in Table 1. For the News CIF resolution sequence, for instance, inserting an I-frame every 30 frames increased the file size (of the 30 fps sequence) by about one-third. There is no motion information associated with an I-frame, however, so the motion file sizes reported in Table 1 would remain the same. Hence, the ratio of motion file size to compressed bit stream size would be lower.

The value of the trade-off between storage space and server side processing could vary depending on the application, lengths of the clips to be streamed, and usage of the server. In scenarios where the clips are relatively short and the number of clients accessing them is high (e.g., news clips on a popular news site) the savings in storage may not be worth the increase in processing. If the clips are really short, video streaming itself may not be needed because downloading the files and then playing them back locally would be feasible. On the other hand, if very long sequences are to be stored on the server, but few clients are expected to access the server concurrently, then the trade-off might be worth it. The four QCIF resolution Coastguard sequences take up 477 kB of space, even though they are only 10 second clips with a single I-frame. Two-hour (7,200-second) versions of these clips would likely take up 343 MB of space. A server with a database of 100 such clips would require 34 GB of storage. The 56% reduction in storage achieved by using the proposed scheme might be an attractive option in this scenario.

4.3.2. Encoding speed

Table 3 provides a comparison between the encoding speeds of the online encoder and the H.263 V3.0 encoder. Data for the V2.0 encoder, which, like the online encoder, uses full search motion estimation, is also included. The data shown is for encoding the six test sequences at 15 fps. For the two original encoders, the input data is the raw file and the output bit stream is written to a local disk. For the online encoder, the input data is the 30-fps-encoded bit stream and the output is sent to the client, which decodes and displays the stream. We have provided the actual time taken to encode each 10-second sequence. The numbers in parentheses indicate the speed in frames per second.

It is observed that the online encoder performs between three to seven times faster than the V3.0 encoder, with the added benefit of using more accurate motion

Table 3. Encoding time comparisons

File	Online encoder	H.263 V3.0	H.263 V2.0
Akiyo QCIF 15 fps	5.5 s (55.0 fps)	31.6 s (9.5 fps)	68.2 s (4.4 fps)
Akiyo CIF 15 fps	18.6 s (16.1 fps)	125.0 s (2.4 fps)	333.0 s (0.9 fps)
News QCIF 15 fps	7.5 s (40.0 fps)	34.1 s (8.8 fps)	78.9 s (3.8 fps)
News CIF 15 fps	25.0 s (12.0 fps)	136.0 s (2.2 fps)	>300.0 s (<1.0 fps)
Coastguard QCIF 15 fps	13.0 s (23.0 fps)	42.3 s (7.1 fps)	103.0 s (2.9 fps)
Coastguard CIF 15 fps	47.6 s (6.3 fps)	176.0 s (1.7 fps)	500.0 s (0.6 fps)

vectors. The V3.0 encoder is itself some two to three times faster than the V2.0 encoder. The performance for the original H.263 encoders would be even slower if one had to decode a 30 fps bit stream and re-encode it at a different frame rate. As one might expect, the performance improvement achieved for the online encoder is greatest for the Akiyo sequence (factor of 5.8 and 6.7 over the V3.0 for QCIF and CIF resolutions, respectively), which has little motion, and least for the Coastguard sequence (factor of 3.2 and 3.7 for QCIF and CIF resolutions respectively), where the motion is high.

These data show that the original H.263 encoders cannot encode the raw video fast enough (on the platform used) for real-time purposes (let alone decode a compressed stream and re-encode it at lower frame rates).

The online encoder, on the other hand, encoded the 15-fps QCIF sequences at 55, 40, and 23 fps for the Akiyo, News, and Coastguard sequences, respectively. This meant that not only were we able to decode a 30-fps bit stream and re-encode it at 15 fps (and of course lower rates) in real time, but we were also able to implement the fast forward control by re-encoding, transmitting, and displaying the lower frame rates at 30 fps (for Akiyo and News, 23 fps for Coastguard). Re-encoding the sequence at 15 fps and displaying it at 30 fps provided a 2× fast forward effect, while re-encoding the sequence at 10 fps and displaying it at 30 fps achieved a 3× fast forward effect and so forth.

However, for the CIF sequences, the online encoder did not exceed 16 fps for any of the three sequences. This meant that only frame rates of 16, 12, and 6 fps and lower could be re-encoded in real time (for the three sequences respectively). By the same token, fast forward control could only be achieved for frame rates less than 16, 12, and 6 fps.

4.4. Limitations

Although real-time operation for CIF sequences is achievable for the current top-of-the-line 1-GHz, or multiprocessor, PCs and workstations, it is clear that the scheme proposed here is not scalable to a server with a large number of concurrent clients, unless very powerful machines (and/or special-purpose hardware) are used. On the other hand, the CIF resolution sequences, are not likely to be used for streaming video over the Internet because they require very high bandwidths that are likely to be used on high-speed corporate Intranets. The three 15-fps CIF sequences used here, for instance, require 94, 123, and 430 kbps, respectively. Most video clips streamed on the Internet today are coded at 160 × 120 resolution (even smaller than QCIF) and in the 20–56 kbps range.

Having said this, even at QCIF resolutions, the proposed scheme cannot scale to very large numbers of concurrent users. As such, it would be more appropriate for moderately used servers (such as those with restricted access).

5. Conclusions

We have proposed a scheme to vary the frame rate of pre-encoded video clips online, particularly in a video streaming application. According to this scheme, a video clip is encoded at high quality and 30 fps and stored on a server. In addition, motion vectors for encoding that same clip at lower frame rates are computed offline and stored on the server. During video streaming, an online encoder can decode the high-quality stream and re-encode it at a lower frame rate in real time (for QCIF and lower resolutions). The additional storage requirements for the motion files are relatively small.

Varying the frame rate provides greater control over the video stream. By reducing the frame rate of the clip, the bandwidth will be reduced without further loss in the spatial quality (temporal quality will be reduced instead). The scheme can also be used to implement a fast forward control: the encoder re-encodes the stream at a lower frame rate, but the decoder plays it back at the natural frame rate. The fast forward effect is thus achieved without increasing the bandwidth consumption significantly or requiring additional buffering at the client side. This technique for providing fast forward control is a compromise between storing multiple versions of the bit stream on one extreme and completely re-encoding the bit stream on the other. It achieves more than 50% savings on storage space compared to the first but is not as scalable in terms of the number of clients supported. It also provides about four or five times improvement in performance over the second, but requires 15–20% more storage.

References

[1] M.-S. Chen, D. Kandlur, and P. Yu, Support for fully interactive playout in a disk-array-based video server, in *Proceedings of ACM Multimedia '94*, San Francisco, CA, pp. 391–398, October 1994.

[2] Ti-Hao Chiang, A rate control scheme using a new rate distortion mode, ISO/IEC JTC1/SC29/WG11 MPEG95/N0436, November 1995.

[3] J. Dey, J. Salehi, J. Kurose, and D. Towseley, Providing VCR capabilities in large-scale video

servers, in *Proceedings of ACM Multimedia '94*, San Francisco, CA, pp. 25–32, October 1994.

[4] A. Eleftheriadis, *Dynamic rate shaping of compressed digital video*, Ph.D Thesis, Columbia University, New YorK, NY, 1995.

[5] L. He, E. Sanocki, A. Gupta, and J. Grudin, Auto-summarization of audio-video presentations, in *Proceedings of ACM Multimedia '99*, Orlando, FL, pp. 489–498, November 1999.

[6] T.-G. Kwon, Y. Choi, and S. Lee, Disk placement for arbitrary-rate playback in an interactive video server, *Multimedia Syst. J.*, 5, 271–281, 1997.

[7] D. LeGall, MPEG: A video compression standard for multimedia applications, *Commun. ACM*, 34, (4), 46–58, April 1991.

[8] A. Srivastava, A. Kumar, and A. Singru, Design and analysis of a video-on-demand server, *Multimedia Syst. J.*, 5, (4), 238–254, 1997.

[9] W. Tavanapong, K. Hua, and J. Wang, A framework for supporting previewing and vcr operations in a low bandwidth environment, in *Proceedings of ACM Multimedia '97*, Seattle, WA, pp. 303–312, November 1997.

[10] Video coding for low bitrate communication ITU recommendation H.263, February 1998.

[11] G. K. Wallace, The JPEG still picture compression standard, *Commun. ACM*, 34, (4), 30–44, April 1991.