FastMPEG: Time-Scale Modification of Bit-Compressed Audio Information

Michele Covell, Malcolm Slaney and Art Rothstein
Interval Research Corporation

ABSTRACT[1]

This paper describes techniques to change the playback speed of MPEG-compressed audio, without first decompressing the audio file. There are two primary contributions in this paper. 1) We describe three techniques to perform time-scale modification in the maximally decimated domain. 2) We show how to infer the output of the auditory masking model on the new audio stream, using the information in the original file. This new FastMPEG algorithm is more than an order of magnitude more efficient then decompressing the audio, performing time-scale modification in the conventional time-domain, and then recompressing. Samples of our results can be found at http://www.slaney.org/covell/FastMPEG/.

1. COMPRESSING AUDIO

This paper describes an algorithm, FastMPEG, to speed up or slow down the playback rate of audio signals that have been bit-rate compressed.

In this paper, we use compression in two different senses. We talk about *time-scale compression* — that is speeding up the playback rate of audio waveforms — and about *bit-rate compression* — that is reducing the number of bits that are used to represent an audio waveform. We preface compression with either time-scale or bit-rate to distinguish these two types of compression. When we talk about *uncompressed audio* or *decompression*, we are always referring to the absence or removal of bit-rate compression.

We describe the FastMPEG algorithms in terms of MPEG layer II compression since all analysis windows are the same length. Extensions to other types of audio bit-rate compression that include subband filtering are easy from this description. We also frame the discussion in terms of speeding up the audio by a factor of 2-to-1. Again, extensions to other (integer or non-integer) time-scale-compression or -expansion rates are easy from this description. Finally, we describe our approach assuming a constant time-scale-compression rate. That is, given 2 packets we hand back 1 packet before receiving more input packets. We assume this, since for many digital AV applications, if this exact output rate is not maintained, it causes the video to stutter. If this constraint is not needed for some application, it can be removed.

There are two primary contributions of this paper. First (Section 2), we describe three different ways to modify the subband streams to time-compress (or -expand) the audio signal. This still requires unpacking the compressed-signal and removing the scaling, but the time-scale modification happens in the maximally decimated filterbank domain. Second, we describe how to recycle the original masking model and to use that model to compress the new signal. Much of the effort needed to compress a signal is used to evaluate an auditory model and infer a model of psychoacoustic masking. In Section 3 we describe how we infer and modify the original masking model with minimal computational effort. Finally in Section 4 we describe our results. First we present a short review of three conventional time-scale compression techniques.

1.1 Time-Scale Compression

There has been a lot of work over the years on time-scale compression: that is, playing back an audio signal at a faster-than-recorded rate. The oldest example of this is the *fast playback* approach. The audio signal is played back at a higher rate by speeding up the analog waveform, shifting the pitch of the signal. The digital equivalent (low-pass filtering followed by subsampling) is also used. With a tape recorder, the fast playback approach has the advantage of extreme simplicity. It has the disadvantage of shifting the pitch of the audio.

Another early form of audio speed up is the *snippet omission* approach [1]. In the analog domain, this is achieved using electromechanical tape players with moving magnetic read heads. These players alternately play and skip short sections of the tape. Again, the digital equivalent (alternately keeping and throwing away short groups of samples) is used. Snippet omission has the advantage of not shifting the pitch of the audio signal. However, it does shift in the frequency domain some of the signal energy, resulting in a pronounced buzzing sound during playback. This artifact is due to the modulation of the input signal by the square wave of the snippet removal signal.

Most recently, there has been a lot of work on the *SOLA* approach (Synchronous Overlap-Add) [2]. SOLA improves on the snippet omission approach, by tying the duration of the segments that are played or skipped to the pitch period of the audio and by replacing the "splicing" with cross-fading. SOLA does not shift the pitch of the signal and it reduces the audible artifacts associated with "snippet omission". It has the disadvantage of

1. Michele Covell's current address is YesVideo.com, 2192 Fortune Drive, San Jose, CA, 95131, covell@ieee.org. Malcolm Slaney's current address is IBM Almaden Research, 650 Harry Road, San Jose, CA 95120, malcolm@ieee.org. Art Rothstein's current address is McKinley Systems, 1757 Ninth Avenue, San Francisco, CA 94122, art@mojo-working.com.

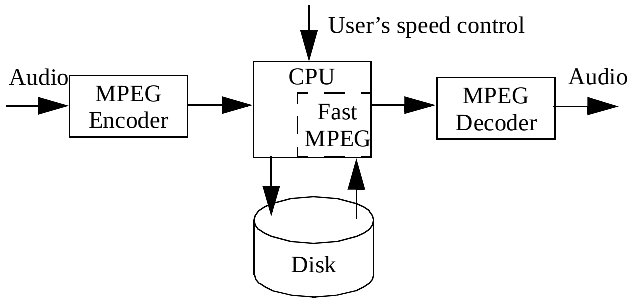

Figure 1: This block diagram shows how FastMPEG fits into a video recorder with a hard disk.

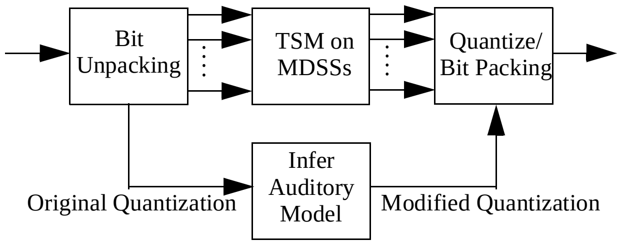

Figure 2: Block diagram of FastMPEG processing steps.

being more computationally expensive, since the amount of overlap that is used is dependent on the local characteristics of the audio signal.

1.2 Bit-Rate Compression

With the proliferation of Internet- and disk-based media application, more and more audio remains bit-rate compressed until after it has been sent to special decompression hardware for playback: the main processor only sees the bit-rate compressed audio. Often in these situations, the main processor has neither the free cycles nor the bandwidth to undo the compression, modify the uncompressed audio, and then recompress the result. An example of such a device is the personal video recorder that allows a TV subscriber to record programs onto disk and to play them back at a later time.

These systems are more useful if they can modify the compressed audio so that it plays back at faster-than-recorded rates, without having to modify the hardware and without having to completely decompress the audio signal. This problem has not been solved by any of the prior work in time-scale modification: earlier work assumes that the audio signal is available in the time domain, in its uncompressed form. We describe three ways to time-scale modify bit-rate-compressed audio, without decompressing. The best solution depends both on the capabilities and availability of the processor that is being used and on the type of audio that has been compressed (speech, music, mixed). A block diagram of a personal recorder and our modifications to the system are shown in Figure 1

In this paper, we frame our discussion in terms of MPEG layer II compression. MPEG layer II compression starts with a bank of 32 filters, each covering 1/32 of the original audio bandwidth. After filtering, each of the subband output streams are subsampled by a factor of 32 and the top 2 bands are discarded. This gives 30 maximally decimated subband streams (MDSSs). The MDSSs are then grouped into packets, with 36 samples from each subsampled subband contributing to the packet (for a total of 36*30 = 1080 samples per packet).Within each packet, each MDSS is "scaled" to bring the maximum value down to a known level and is quantized with a fixed number of bits assigned to each sample. The MPEG algorithm then packs the scaling factors and the quantized samples into a bit stream.

2. MODIFYING THE WAVEFORM

There are four steps in the FastMPEG algorithm; they are shown in Figure 2. First we unpack the MPEG bit stream, remove the scaling, and separate the 30 MDSSs. Second, we perform the same time-scale modification procedure in parallel on all 30 MDSSs. Third, we infer and modify the psychoacoustic masking model. Finally, we scale the subband streams, requantize the new signals with the recycled auditory model, and pack the bits into a new MPEG bit stream. In this section we describe how the three different approaches to time-scale modification described in Section 1 are modified to make them applicable to MDSSs.

Section 2.1 TSM using snippet omission approach

The first, and simplest, way to achieve time-scale compression is the snippet omission approach. Upon receiving two packets and undoing their scaling and quantization, we take 72 samples (two packets) per MDSS and simply drop groups of samples from each MDSS. If N is the snippet length, we start a new output stream by saving, into the output packet, the first N samples per MDSS. After this, we then skip over the next N samples per MDSS. We repeat this until we reach the end of the 72 input samples in the two MDSS packets. Typically, N will be chosen so that it is a divisor of 36 (usually, N = 6, 9, or 12). If this is not the case (if N is not a divisor of 36), then we need a slightly different algorithm on output packets after the first one, to compensate for this uneven division.

Section 2.2 TSM using fast playback approach

The second way to modify the time scale is the fast playback approach. With the digital version of fast playback for time-scale compression, the frequency domain structure of the audio signal changes so that only the bottom half of the original spectrum is retained and this bottom half expands linearly to cover the full range from zero to the Nyquist rate. The algorithm simulates this frequency shifting behavior but it does so in the maximally decimated frequency domains of the subband streams.

Upon receiving two packets, we throw away all of the samples from the MDSSs that correspond to the upper half of the frequency bands, without undoing their scaling or quantization. We undo the scaling and quantization on the remaining half of the MDSSs, those that correspond to the lower frequency bands. We take the samples from these MDSSs, low-pass and high-pass filter each, downsample by a factor of two, and redistribute the low-pass and high-pass outputs to the full set of MDSS.

To make this algorithm more clear, consider the ith MDSS, where $0 <= i < 15$. We low-pass the 72 samples corresponding to the ith MDSS and downsample by 2, giving 36 samples. We assign these 36 samples to the $(2*i)^{th}$ MDSS in the output packet. We high-pass the same 72 input samples (corresponding to the ith MDSS) and downsample by 2, giving 36 samples. We assign these 36 samples to the $(2*i+1)^{th}$ MDSS in the output packet.

If computational resources are limited, these "low-pass" and "high-pass" filters can be as simple as 2-point sums and differences (followed by division by 2, to maintain the overall power levels).

Section 2.3 TSM using SOLA approach

Our most expensive way to time compress bit-rate-compressed audio is to use the SOLA approach. Upon receiving two packets and undoing their scaling and quantization, we use one of the frequency bands to determine the optimal snippet length. For most applications, we simply use the lowest frequency band for this determination. However, if we are working with a band-limited audio input signal (e.g., telephone speech), we instead check to see which MDSS has the maximum energy across the two input packets and we use that MDSS to determine the correct snippet length.

Once we have selected a MDSS to analyze, we use that channel's autocorrelation to determine the optimal snippet length. For computational efficiency, we do this by zero-padding the 72 points out to a length of 128 points, taking a 128-point real-input FFT, replacing the FFT values with their magnitude squared values, then taking a 128-point real-symmetric-input IFFT. The output of this process is a (real-valued) 128-point function that corresponds to the auto-correlation of the original 72 input points, with the zero-lag index as the first sample and with the largest-lag indices temporally aliased around the 64th sample. We can allow this temporal aliasing, since we know that we will not be using the largest lags as our snippet length.

We find the highest autocorrelation peak after the peak at zero lag. The index of this peak (which we will continue to refer to as N) gives us the snippet length that we should use in this pair of input packets. We only do this part of the process (the computation of the autocorrelation and the peak picking) on one of the 30 MDSSs. We use whatever snippet length that we get from this one channel on all of the other channels.

Once we have found N, we can construct the output packet, using the same approach as described above for snippet omission. Unlike SOLA in the uncompressed domain, we do not cross-fade to create the final output packet, but instead simply omit snippets. We do not cross fade since our estimate of the pitch period is heavily quantized. Cross fading using a heavily quantized pitch-period estimate degrades the audio quality (as well as increasing the computational requirements)**.**

It is worth noting one important difference in the result of autocorrelation as used by SOLA in the uncompressed and maximally-decimated domains. Conventionally, during voiced speech,

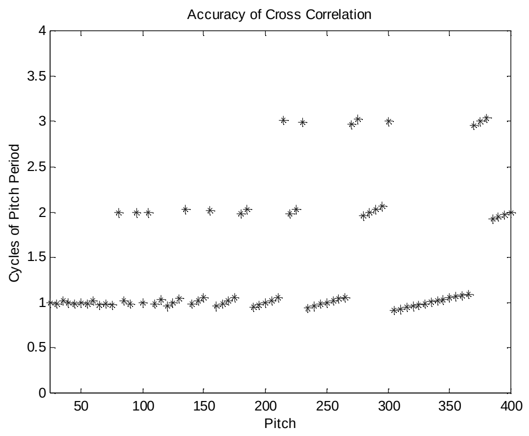

Figure 3: Accuracy of snippet length estimation as a function of pitch. If we knew the exact pitch and used the closest downsampled integer shift in the SOLA algorithm, then the standard deviation of the error is 18.57% of a period. On the other hand, the standard deviation of the error with the downsampled integer shift using the first peak of the autocorrelation is 2.2% of the pitch period.

autocorrelation produces a peak at the pitch period. The first peak of the autocorrelation provides a good estimate of the periodicity of the signal, without introducing a large snippet length during which the signal might be changing. In the maximally-decimated domain of the MDSS, the pitch period might not fall near an integer number of samples. The maximum of the autocorrelation function often is a multiple of the pitch period. This is shown in Figure 3: the location of the highest point in the autocorrelation is at different multiples of the period as the pitch is changed. The preferred snippet length changes from one pitch period to two or more periods, thereby introducing large jumps in the snippet length, even when the pitch is varying smoothly. Using multiples of the fundamental results in better sounding audio than we would get by trying to force the snippet length to the nearest (decimated) quantization of the fundamental period. The reason for the better result is that the multiple-period snippet removal more closely approximates removing an integral number of pitch periods.

3. RECYLING THE AUDITORY MODEL

Two steps are needed to finalize the compression of the modified audio signal. First, we need to calculate a new psychoacoustic masking model and the appropriate quantization levels to preserve the quality of the audio signal. Then we adjust the quantization levels for each band and packet so that we generate an MPEG bit-stream with the original bit-rate.

MPEG layer II, as well as most other audio compression algorithms, includes a psychoacoustic model, to determine how many bits to allocate to each subband. This psychoacoustic model is used to predict the amount of quantization noise in each frequency band that is masked by the signal content. By using a psychoacoustic model, the compression system keeps the perceived quality of the audio as high as possible for the given number of bits used to encode it.

These psychoacoustic models are expensive to compute. Ideally we would recompute the masking model after we perform the TSM algorithm. Fortunately, the new signal is much like the original signal and instead we recycle the auditory model.

The original MPEG bit stream does not explicitly include the masking levels used to determine how to quantize the original audio, but it does include the number of bits used to quantize each subband. We therefore use the original bit allocations of the input packets to determine the bit allocation of the output packet. We implicitly infer the new masking model from this evidence we have about the original model. The new masking model is not exact, but, arguably, the artifacts introduced by any of the three TSM algorithms are much larger than the errors due to estimating the auditory model from the quantization levels.

We estimate the new auditory model by computing the band-by-band maximum of the quantization levels used in the packets we combined to form the new packet. Thus if we are using SOLA to compress two packets into one, the band-by-band maximum of the quantization levels of these two packets is used as the desired quantization level of the new packet. On the other hand, if we are using fast playback then the maximum of the quantization levels for subband i in the two packets is assigned to subbands $2*i$ and $2*i+i$.

With this process we find the desired quantization levels for each subband and each packet. Unfortunately, the total bit budget, the sum of the bits used to quantize each band, is limited. Furthermore, in fast playback, where low-frequency bands are shifted to high-frequencies, there are fewer legal quantization levels (see Table B.2 of the MPEG audio standard [3]).

We use an iterative procedure to determine the final bit rate. Starting with the desired bit rate, we round an illegal subband's bit rate up to the next higher legal bit rate. If the cumulative rate is less than or equal to the allowed bit rate, then we are done. Otherwise, starting at the highest frequency subband, we reduce the desired bit rate by one, if necessary increasing the desired bit rate up to the next higher legal value and sum the needed bits to see if we are at the allowed rate. We reduce the desired rate by one bit across all the bands before restarting at the highest subband. This process is similar to the water-filling approach used to define the original allocation, but is quicker since we expect the new allocation to look much like the original.

At this point the time-scale modified audio can be scaled, quantized and packed into an MPEG stream.

4. RESULTS

The FastMPEG algorithm is an efficient way to modify the time-scale of a data-compressed audio signal. To quantify our efficiency, we compare the cost of our (unoptimized) implementations running on a 333MHz Pentium II processor. Unpacking the bits, rescaling and repacking the bits (the null operation) takes 2.9% of real time. Encoding the sound with version 1.0 of the Linux utility wav2mp takes 72.6% of real time.

The audio signals produced by FastMPEG are degraded by both the TSM approach and the assumptions that are broken by modifying the critically-sampled filterbank approach. There are also artifacts introduced by when we recycle the auditory model, but we believe the TSM and resulting filterbank errors swamp the errors due to masking. Examples of FastMPEG's results are available on paper's web site (http://www.slaney.org/covell/FastMPEG/).

The audio quality for the snippet omission approach suffers primarily due to the spectral energy splattered by the chopping action. This modulated energy violates the assumptions behind the perfect-reconstruction filterbank, but the overall quality is similar to the quality of the snippet-omission approach on the uncompressed waveform. The primary advantage of this approach is its low computational complexity. This approach takes 2.2% of real time.

With fast playback, the audio quality will depend, to some extent, on the filters used to separate the input bands into two output bands. We used inexpensive 2-point filters, which introduces aliasing into the result. Even so, the most noticeable perception of fast-playback modified audio is the change in pitch. Through all the FastMPEG stages, the resulting audio is clean and, other than the pitch change, the artifacts are small. The primary advantage of this approach is its plausible result — modified but understandable — with relatively small computational effort. This approach takes 3.0% of real time.

Finally, the best audio quality comes from the SOLA approach applied to FastMPEG. SOLA applied to uncompressed audio produces very clean audio — artifacts are generally imperceptible to untrained listeners. Unfortunately, the SOLA operation violates the perfect reconstruction assumptions and this causes small artifacts. The result is best described as similar in quality to AM radio. Like SOLA in the uncompressed domain, this algorithm works best on audio with a single speaker, or single pitch at any one time. This approach takes 3.3% of real time.

We have described a system for time-scale modification of bit-compressed audio. These techniques will become more valuable as more audio is compressed for delivery to inexpensive devices

REFERENCES

[1] Paul Gade, Carol Mills. "Listening Rate and Comprehension as a Function of Preference for and Exposure to Time-Altered Speech." *Perceptual and Motor Skills,* vol. 68, pp. 531–538, 1989.

[2] S. Roucous, A. M. Wilgus. "High Quality Time-Scale Modification for Speech." *IEEE International Conference on Acoustics, Speech, and Signal Processing,* vol. 2, pp 493–496, Tampa, FL, 1985.

[3] ISO/IEC JTC 1, 1993. "Information Technology - Coding of moving pictures and associated audio for digital storage media at up to about 1,5 Mbit/s - part 3: Audio, Technical Corrigendum 1", International Standard ISO/IEC 11172-3, pp. 46–49.