A LOSSLESS AUDIO COMPRESSION SCHEME WITH RANDOM ACCESS PROPERTY

Dai Yang[1], *Takehiro Moriya*[2], *Tilman Liebchen*[3]

[1]Univ. of Southern California, Dept. of Electrical Engineering, Los Angeles, CA, USA
[2]NTT Cyber Space Laboratory, Media Processing Project, Musashino, Tokyo, Japan
[3]Technical University Berlin, Communication Systems Group, Berlin, Germany
daiyang@alumni.usc.edu, moriya.takehiro@lab.ntt.co.jp, liebchen@nue.tu-berlin.de

ABSTRACT

In this paper, we propose an efficient lossless coding algorithm that not only handles both PCM format data and IEEE floating-point format data, but also provides end users with random access property. In the worst-case scenario, where the proposed algorithm was applied to artificially generated full 32-bit floating-point sound files with 48- or 96-kHz sampling frequencies, an average compression rate of more than 1.5 and 1.7, respectively, was still achieved, which is much better than the average compression rate of less than 1.1 achieved by general purpose lossless coding algorithm gzip. Moreover, input sound files with samples' magnitude out-of-range can also be perfectly reconstructed by our algorithm.

1. INTRODUCTION

Current state-of-the-art ”perceptually lossy” forms of audio encoding, such as Dolby AC-3 and the ISO/MPEG audio standards, are capable of achieving compression ratios up to 12:1 and higher, and have thus become popular as components of Internet media applications, mobile terminals, and special-purpose audio-visual data-storage devices. However, lossy audio-compression algorithms have met with strong resistance from the fields of professional studio operations, sound archiving, and the disc-based consumer market. This is because a lossy compression algorithm, regardless of how good it is, permanently alters the original recorded sound data and thus inherently reduces audio quality. In contrast, copies of an audio waveform reproduced after lossless audio compression are always bit-by-bit identical to each other, no matter how many cycles of compression and decompression are applied.

In October 2002, MPEG issued a new call for proposals on MPEG-4 lossless audio coding [1]. The new standard is now being developed and will be finalized in a year or two. However, most of the available lossless audio coding algorithms focus on audio input sources of PCM format. Little work has been done on audio sources of IEEE floating point format [2]. As an important sound format in the audio industry, IEEE floating point sound files have the ability to store the audio signal much more precisely than sound files with regular PCM format, and are therefore much more difficult to compress losslessly.

In our previous work [3], we described a lossless compression scheme for audio sources with IEEE floating point format. In that paper, the floating point input file is first converted into regular PCM data, then the PCM data and the floating point residue are taken care separately. The core coding module adopted there to compress the PCM data is Monkey's Audio [4], which has no random accessibility at all. Since random access not only gives the end users more control of the compressed bitstream but also limit the length of possible error proporgation when transmitting the bitstream through networks, based on our previous work, we developed a new lossless audio coding technique that not only handles regular PCM data and floating point data but also provides end users with random access property. In some circumstances, after sound files are edited and manipulated, the final data may contain out-of-range samples. By using the proposed algorithm, these kinds of files can also be perfectly reconstructed.

The rest of this paper is organized as follows. Section 2 describes the proposed algorithm in detail. Experimental results and some discussions are provided in section 3. Finally, some concluding remarks are given in section 4.

2. COMPLETE ALGORITHM DESCRIPTION

2.1. System Overview

Figures 1 and 2 show block diagrams of the encoder and decoder, respectively. On the encoder side, audio samples are buffered and processed frame by frame. Each frame is divided into blocks of samples. Typically there is one block for each channel. Input data with out-of-range magnitude are normalized before further processed. Depending on the input file format, data is sent either directly into the lossless PCM coding module or floating point to PCM conversion module. After converting floating point data to PCM format, the generated PCM data and their corresponding floating point residue are compressed separately. Thus, for floating point sound file, its bitstream is a combination of compressed PCM data and floating point residue.

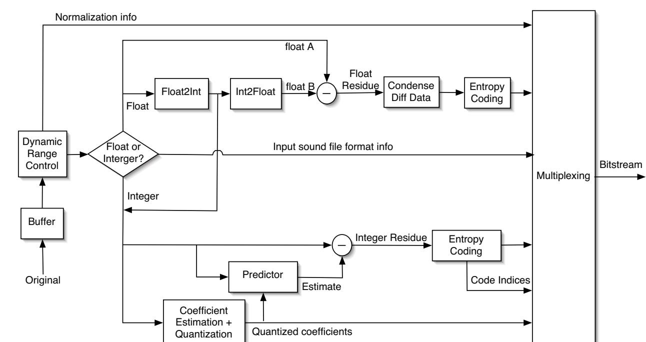

Fig. 1. Encoder block diagram.

0-7803-8484-9/04/$20.00 ©2004 IEEE

In March 2003's MPEG meeting, the lossless PCM coding module proposed by Technical University of Berlin [5] has been accepted as one of MPEG-4 lossless audio standard's first reference models. Because of its excellent coding performance, it is adopted in our lossless PCM coding module in this work. The PCM coding module consists of three major building blocks, i.e. coefficient estimation and quantization, predictor, and entropy coding of integer residue. The basic version of the PCM encoder uses one sample block per channel in each frame. Optionally, each block can be subdivided into four shorter sub-blocks to adapt to transient segments of the audio signal. The encoder generates bitstream information allowing random access at intervals of several frames. Furthermore, joint stereo coding can be used to exploit dependencies between the two channels. For each channel, a prediction residual is calculated using linear prediction with adaptive coefficients and adaptive prediction order. The coefficients are quantized prior to filtering and transmitted as side information. The prediction residual is entropy coded using one of several differing Rice codes. The indices of the chosen codes have to be transmitted.

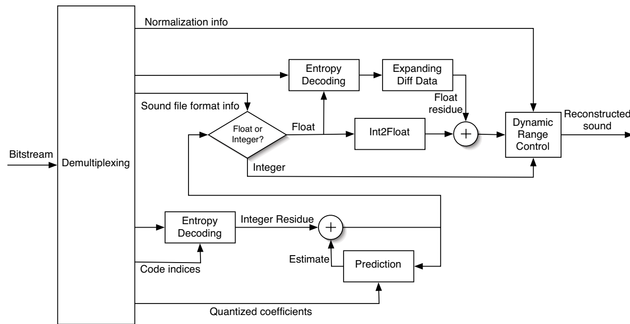

Fig. 2. Decoder block diagram.

For floating-point input files, their samples need to be first truncated to integer. The floating-point residue data is generated byte-wisely by finding the difference between the original normalized floating-point samples A and the truncated integer samples in floating-point format B. In this way, any possible calculation error caused by simple floating-point domain subtraction can be avoided so that perfect reconstruction at the decoder side can always be guaranteed. After eliminating unnecessary bits, the floating-point residue data is then packed and entropy coded. A multiplexing unit finally combines all coded bits from the PCM encoder, floating-point residue encoder and other side information bits to form the compressed bitstream. The encoder also provides a CRC checksum, which is supplied mainly for the decoder to verify the decoded data. On the encoder side, the CRC can be used to ensure that the compressed file is losslessly decodable.

The decoder is significantly less complex than the encoder. It first decompress the entropy coded integer residual and then using the predictor coefficients to calculates the lossless integer signal. If the original sound file is in floating-point format, the decoder also needs to decompress the floating-point residue and add them to the integer part. The normalization information is used at the end to restore audio data's original dynamic range.

2.2. Bitstream Structure

The general bitstream structure of a compressed file is shown in Figure 3. The bitstream consists of a header, compressed information of super-frames, any possible non-audio data, and a CRC checksum field. The header consists of the actual file header, followed by the header of the original sound file. Currently, the encoder only supports PCM wave files (*.wav) as input files, and the wave header is directly embedded in the data stream of the compressed file. Each super-frame is a random access unit. The field "R" appears at the beginning of each random access unit (e.g. each M frames) and specifies the distance (in bytes) to the next random access unit. Any prediction related parameters need to be reset at the boundary of each random access unit. Remaining non-audio bytes of the wave file are embedded after the last audio frame. The CRC checksum is stored at the end of the compressed file. If non random access mode is selected by the users, the compressed data will be cascaded one frame after the other and put between the header and non audio data part. In addition, no "R" field is necessary at the beginning of each frame data.

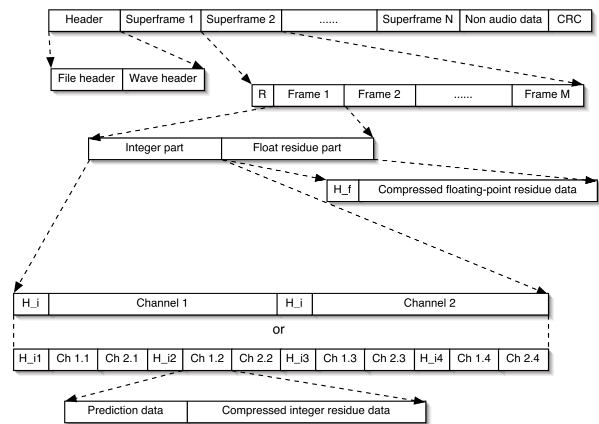

Fig. 3. Bitstream architecture.

For PCM input files, each frame only contains an integer part. For floating-point input files, there is a floating-point residue part following the integer part. The integer part of each frame consists of one or four sample blocks for each channel, where each block has its own block header "H", carrying general information about the block (e.g. silence block, joint stereo difference block, etc.). The block itself typically contains the prediction data and the compressed integer residual values. The floating-point residue part contains a header and compressed floating-point residue data. The floating-point part header contains necessary information to decode the floating-point residue. The residue data is a compressed version of the exponent and mantissa difference of each sample.

2.3. Byte-wise Difference

Instead of doing a floating-point subtraction, we generate the difference between the floating-point samples A and B byte-wisely so that perfect reconstruction can always be guaranteed. Figure 4 illustrates how the byte-wise difference is taken for one data sample. In our implementation, there is no need to record the difference of the sign bit because all sign bits are kept the same after format conversion. The exponent and mantissa differences, denoted

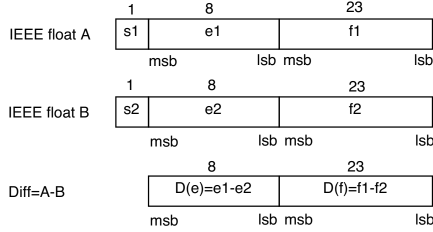

Fig. 4. Byte-wise difference.

by $D(e)$ and $D(f)$, consist of 8 and 23 bits, respectively. They can be calculated either by bit-wise XOR or by a simple subtraction. Each sample's difference data is stored in a 32-bit format, leaving one reserved bit for any possible need in the encoding process.

2.4. Format Conversion

How the PCM and floating-point format conversion is handled has a great influence on the generation of floating-point residue data, which greatly affects the overall compression performance.

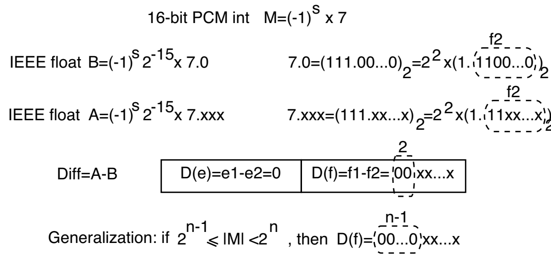

Fig. 5. An example of the new format conversion.

In most existing wave file format conversion programs, such as [6], floating-point numbers are rounded up/down to the nearest integer when IEEE floating-point samples are converted to PCM format. Doing so minimizes the magnitude of the residue. However, the sign of each sample's residue could be positive or negative. Moreover, exponents of some samples' residues could be nonzero, which makes the compression of residue data inefficient and thus degrades the overall coding performance. In the proposed compression system, the magnitude of the floating-point number f is truncated to the largest integer number that is less than or equal to f, i.e., $|f| \to \lfloor |f| \rfloor$. When comparing corresponding floating-point samples in IEEE files A and B, i.e., f_A and f_B, the advantages and disadvantages of doing so over the traditional rounding method can be summarized as below:

Disadvantage:

- The average magnitude of the residue is increased.

Advantages:

- Exponents remain unchanged, unless truncation to zero occurs.
- Signs of non-zero residues are always the same.
- The number of non-zero bits in $D(f)$ can be calculated implicitly.

An example of how to calculate the number of nonzero bits in $D(f)$ is shown in Figure 5. Suppose that a sample M in the intermediate PCM file has its magnitude equal to 7. When converted to IEEE 32-bit floating-point format, it has the form of $(-1)^s 2^{-15} \times 7.0$, where 7.0 has the binary representation of $(111.000...0)_2$. If the sample M is converted by our new format conversion method, its corresponding floating-point sample A should have the form of $(-1)^s 2^{-15} \times$ 7.xxx...x, where 7.xxx...x $\geq$ 7.0 and has the binary representation of $(111.\text{xxx...x})_2$. When the byte-wise difference is taken, we notice that the highest two bits in $D(f)$ are zeros. At the decoder side, given a sample value in the intermediate PCM file, the number of nonzero bits in $D(f)$ can also be calculated. Therefore, only those nonzero bits of $D(f)$ need to be sent to the entropy coder for further processing. On the other hand, if the traditional format conversion is performed, the number of nonzero bits is not predictable, and all 23 bits [2] of $D(f)$ need to be passed to the entropy coder. This simple example can be generalized into the following rule.

Proposition *If a sample in the intermediate PCM file has its magnitude greater than or equal to 2^{n-1}, but smaller than 2^n $(n > 0)$, then the highest $n-1$ bits in $D(f)$ should always be zero.*

2.5. Handling the Floating-point Residue

For floating-point input files, the total bitstream is the combination of integer part I and floating-point residue part D, where I is usually 15% to 30% of the file size of the original 32-bit floating-point file. Since the intermediate raw residue data is the same size as raw data in original floating-point file, it is impossible to achieve a satisfying compression result if the residue data are poorly handled.

Using the new format conversion method, most of the samples' $D(e)$ equal zero. Therefore, there is no need to send these $D(e)$ to the bitstream. As for these samples' corresponding $D(f)$ values, only those possible nonzero bits need to be buffered and sent to the entropy coder. When a sample in the original floating-point file has its magnitude smaller than 2^{-15}, zero truncation must be done when converting this number into a 16-bit integer. Thus its corresponding $D(e)$ is nonzero and $D(f)$ may have all 23 nonzero bits.

Our new format conversion method can calculate the number of highest zero bits of $D(f)$. However, the remaining lower bits of $D(f)$ may still contain several significant zero bits. One extreme example is when the input floating-point file is a simple format conversion from a PCM file without any gain factor involved. In this case, all floating-point residues are zero. In order to improve the performance for input files of this particular kind while maintaining similar performance for other input files, the highest nonzero byte or bit in a whole block, $N_{high-block}$, is calculated and sent to the bitstream. When compressing $D(f)$, only bits that are lower than the smaller value of $N_{high-block}$ and N_{sample} will be processed, where N_{sample} is the number of possible nonzero bits calculated by the new format conversion method for each sample.

The entropy coder used to compress the compacted floating-point residue data is the general purpose byte-wised lossless coder gzip. In order to achieve the best coding performance, a selection procedure based on the empirical entropy of the residue data

Table 1. Compression results

Sfreq	G/B	algorithm	RA (ms)	Avemaria	Clarinet	Cymbal	Etude	Flute	Haffner	Violin	Average
48kHz	1.0/16	ours	500	5.106	4.195	6.772	4.729	4.929	3.616	4.100	4.778
			no	5.109	4.197	6.779	4.731	4.932	3.617	4.102	4.781
		gzip	no	1.774	1.735	2.708	1.730	1.890	1.590	1.789	1.888
	2.99/24	ours	500	1.591	1.509	1.478	1.575	1.514	1.499	1.468	1.519
			no	1.591	1.509	1.478	1.575	1.514	1.499	1.468	1.519
		gzip	no	1.083	1.082	1.175	1.081	1.084	1.079	1.081	1.095
96kHz	1.0/16	ours	500	7.155	6.714	8.464	6.636	7.422	5.822	6.441	6.951
			no	7.159	6.717	8.470	6.639	7.426	5.825	6.444	6.954
		gzip	no	1.793	1.749	2.801	1.746	1.907	1.600	1.806	1.915
	2.99/24	ours	500	1.762	1.831	1.531	1.757	1.761	1.819	1.757	1.745
			no	1.762	1.831	1.531	1.756	1.761	1.817	1.757	1.745
		gzip	no	1.803	1.803	1.177	1.082	1.085	1.079	1.082	1.096

is enforced, so that either the original or the byte-aligned version of the compacted floating-point residue data is input into the gzip encoder. In this way, we experienced a significant performance improvement when the input floating-point file is a simple format conversion of a PCM file.

3. EXPERIMENTAL RESULTS

The proposed compression scheme has been implemented and tested. We evaluated the performance of the proposed algorithm in the worst-case situation by applying it to IEEE 32-bit floating-point format audio source files, which were artificially generated by multiplying PCM sound files by different gain factors. Table 1 shows compression results for two sets of seven input sound files with sampling frequencies of 48- and 96-kHz, respectively. The number pair G/B in the second column, e.g. 2.99/24, means the IEEE floating-point input file is generated by multiplying the 24-bit PCM file by a gain of 2.99. In this table, we list compression rates of 500 ms random access mode and that of non random access mode. In order to demonstrate the compression efficiency of the proposed algorithm, we also list the compression rates by using general purposed lossless compression engine gzip. From the values in this table, we observe the following points:

- When compared with non random access mode, in most of cases, random access mode only slightly degrades the coding performance by inserting the ”R” field and resetting the prediction parameters.
- The proposed compression scheme has better performance for input files with a 96-kHz sampling frequency than for 48-kHz ones.
- When compared with gzip, the non random access mode of the proposed algorithm achieves significantly better results, especially when the input files are generated with parameter G/B=1.0/16, which is generally true for any input file that is a simple floating-point conversion of a PCM file.

4. CONCLUSION

We proposed a new lossless compression scheme with random access property for audio data either in the IEEE 32-bit floating-point format or regular PCM format. The proposed algorithm utilizes an prediction-based PCM lossless audio coder as a core coder, then gracefully handles the floating-point residue data to generate the final bitstream. In the worst case scenario, where the input IEEE floating-point files are generated by multiplying the 24-bit PCM file by a non-integer number, the proposed algorithm still achieves an average compression rate of more than 1.5 and 1.7 for input files with sampling frequencies of 48- and 96-kHz, respectively. When the input floating-point file is a simple format conversion of a PCM file, a compression rate of nearly 5.0 (48-kHz) or 7.0 (96-kHz) can be achieved. Moreover, input sound files with samples' magnitude out-of-range can also be perfectly reconstructed by our algorithm.

5. REFERENCES

[1] ISO/IEC JTC 1/SC29/WG11 N5208, *Revised call for proposals on MPEG-4 lossless audio coding*, Shanghai, China, October 2002.

[2] ANSI/IEEE Std. 754-1985 American National Standard, *IEEE Standard for Binary Floating-Point Arithmetic*, New York, 1985.

[3] D. Yang and T. Moriya, "Lossless compression for audio sources with IEEE floating point format," *AES 115th Convention*, October 2003.

[4] Monkey's Audio, *A fast and powerful lossless audio compressor*, http://www.monkeysaudio.com.

[5] T. Liebchen, "Mpeg-4 lossless coding for high-definition audio," *AES 115th Convention*, October 2003.

[6] P. Kabal, *AFsp Library, programs and routines*, http://www.tnt.uni-hannover.de/soft/audio/packages/afsp.

[7] M. Hans and R. W. Schafer, "Lossless compression of digital audio," *IEEE Signal Processing Magazine*, vol. 18, no. 4, pp. 21–32, 2001.