CCRMAC-3 Perceptual Audio Coder

Graham Davis, Mike Olsen, Shenli Yuan
Stanford University - CCRMA
Music 422 - Professor Marina Bosi

I. MOTIVATION

The goal of a high quality perceptual audio coder is to reduce the encoded audio data rate as much as possible, while maintaining a high fidelity and perceptually identical decoded audio signal. The previously implemented CCRMAC-2 base coder succeeds in this task at data rates of approximately 192 kbps on a select few audio samples. However, reducing the data rate and coding critical audio samples quickly displays the shortcomings of CCRMAC-2. At 128 kbps, CCRMAC-2 is no longer in the region of transparency due to clearly audible coding artifacts. Furthermore, at 192 kbps, coded signals with sharp transient attacks display high levels of pre-echo. In an effort to improve the perceptual transparency of CCRMAC-2 at lower data rates over all critical audio samples we propose the development of the CCRMAC-3 perceptual audio coder. In what follows, we discuss the added features of CCRMAC-3, their implementation and the resulting benefits.

II. OVERVIEW

The CCRMAC-3 perceptual audio coder relies on three new features to optimize bit allocation and better handle signal transients. The three new features are:

- Entropy (Huffman) Coding
- Bit Reservoir
- Transient Detection and Block Switching

Huffman coding is a method by which the quantized block floating point mantissa values are represented in the fewest number of bits possible. Utilizing six Huffman Tables (each trained on a different critical audio sample), we determine the best case scenario for coding each block of mantissa values. Huffman coding parameters (e.g. which Huffman Table was used) are transmitted in the encoded file to allow for accurate decoding. This implementation will be discussed further in the following section.

The coding gains per block produced by Huffman coding are tracked in a bit reservoir, and used in future blocks. While a constant global data rate is maintained, this locally variable data rate allows for higher bit allocation to individual bit starved blocks.

Transient detection and block switching allow for better handling of onsets within a critical audio file. Without block switching, transients occur within the context of a block with length much greater than the transient itself. Coding artifacts from the transient are spread throughout the block, leading to pre-echo prior to the onset of the transient. The effects of pre-echo are most noticeable in the CCRMAC-2 encoded castanet and glockenspiel audio files. In CCRMAC-3 we implemented a transient detection module, which alerts the encoder when the following signal block includes a transient. When transients are present, the encoder switches to a shorter MDCT block size, allowing for higher temporal accuracy. This cuts down on the spreading of coding artifacts around transients in the time domain, removing pre-echo from the decoded audio signal.

In the following section, we take a deeper look into the implementation of these three features and how the CCRMAC-2 base coder was modified to support them.

III. IMPLEMENTATION

A. Huffman Coding

Background

Huffman coding is a classic entropy coding method that can increase the compression ratio by using fewer bits to represent the more frequently occurring data symbols. Based on the different frequencies that different data symbols occur, Huffman coding assigns variable-length codes: the more frequently occurring symbols will be assigned shorter codes while the longer codes are assigned to less common symbols.

Standalone Huffman Module

The Huffman encoding was first implemented as a standalone module which is able to encode and decode data, generate new Huffman table and Huffman tree, as well as building Huffman table and Huffman tree based on existing tables and trees.

The workflow of the standalone Huffman module is as follows :

1. Generate frequency table
2. Build Huffman tree: used for decoding
3. Build encoding map: used for encoding
4. Encode the data according to encoding map (Huffman tree)
5. Decode the data according to the Huffman tree.

The first step of the implementation is to build the frequency table, which is essentially a map between different data symbols and their times of occurrence. The frequency table is

designed in a way that we can input an old frequency table, and update the values according to the new data input. In this way the user is able to reuse the old data, or train the table with multiple datasets, which saves a significant amount of time and computer resources.

The frequency table is then sorted according to the frequencies. For example, if we have data “abbbbbcccdeeeeff”, the sorted frequency table will be {‘a’: 1, ‘d’: 1, ‘f’: 2, ‘c’: 3, ‘e’: 4, ‘b’: 5}, which is a priority queue. After that, the first two elements in the priority queue will merge into one node of binary tree, with the two elements being the left and right branches of the node respectively. This process will keep going until there is one element left in the priority queue, which is also the root of the Huffman tree. The generation of Huffman tree for the example input data is shown in figure 1 below. The box at the right hand side represents the complete Huffman tree.

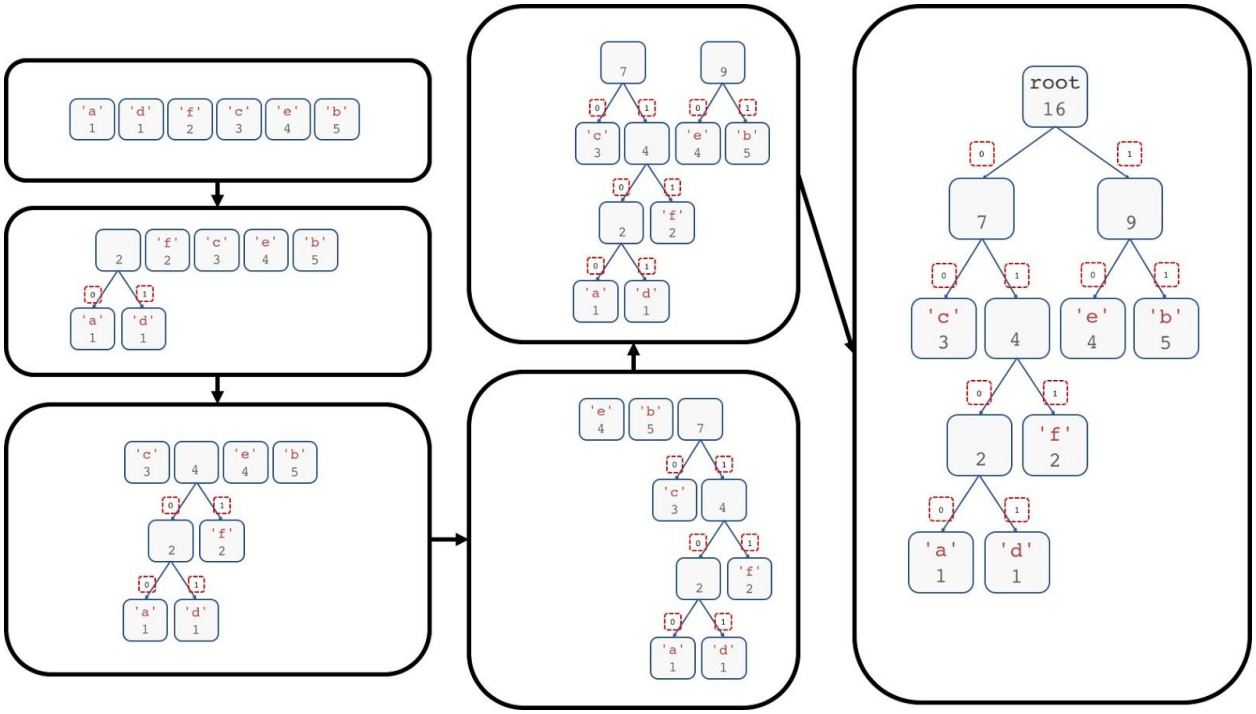

Figure 1. Example Huffman Tree generation

The next step is walking down the Huffman tree to build the Huffman table (encoding map) used for encoding data. Each leaf of the Huffman tree represents one data symbol, and the code assigned to the symbol can be obtained by walking down the Huffman tree from the root to the symbol. In this example, symbol ‘c’ has the code ‘00’, the symbol ‘a’ has the code ‘0100’, the symbol ‘d’ has the code ‘0101’, etc. After getting the codes for all the data symbols, we will have a full Huffman table.

Huffman in CCRMAC-3

Our Huffman module is used to encode signal mantissa values in the CCRMAC-3 perceptual audio coder. Due to data constraints, our implementation does not pass the Huffman table and Huffman tree through the data stream. We trained 6 sets of Huffman tables and Huffman trees with different sound genres (step 1 - 3). We then store these Huffman tables and trees in the encoder and decoder respectively, to be used in the encoding and decoding process (step 4 - 5). Because we are only encoding the mantissas (maximum 16 bits), to avoid the situation where the data does not exist in the pretrained Huffman tables and Huffman trees, in the training we include all the possible values for 16-bit data, and assign the weight of 1e-16 for any data that does not show up in the training data. In this way no matter what data eventually show up in the block, we will have a corresponding code in the Huffman tree and Huffman table. The reason for assigning 1e-16 weight to these data is to avoid an excessively deep tree. As we can envision, if these data are assigned with a weight of 0 during the generation of the Huffman tree, the node will also be at the front of the priority queue. As a result the tree will grow much deeper in the vertical direction. This will greatly increase the coding length of these weight 0 codes. Including this initial weight for the no-show data, allows for a more balanced tree with fewer maximum length codes. This method fulfills the task of an "escape code" while at same time maintaining the elegance of the Huffman method. Our Huffman coder works as a stand alone module, which was then embedded in the CCRMAC-3 coding process.

During the encoding process, CCRMAC-3 begins by reading a block of raw PCM data. The PCM mantissa values are quantized using a block floating point quantizer. The quantized mantissas are then encoded using all 6 of the previously trained Huffman tables. The overall bit length of the 7 forms of quantization are compared, and the optimal representation is chosen. In this way, CCRMAC-3 only uses Huffman coding when it provides coding gains. Upon writing to the PAC file, CCRMAC-3 passes a 3-bit Huffman Table code in the header of each data block. This code is used by the decoder to determine which Huffman tree to use in the decoding process (a value 000 indicates that no Huffman coding was used). The mantissa values of a Huffman encoded block are preceded by a 10-bit Huffman length term. This value is used by the Huffman decoder to accurately read and decode a stream of Huffman encoded bits. The bits used for this term are taken into account in the calculation of the optimal quantization.

To decode, CCRMAC-3 begins by reading the 3-bit Huffman table index of a PAC file data block. If this value is 000, the block is decoded as a normal quantized block of mantissas. For all other Huffman table values, the mantissas are first decoded by the Huffman module. The first 10-bits of the mantissa block are read as nHuffBits, and the following nHuffBits-bits are read as the Huffman encoded mantissa block. These values are decoded by the correct Huffman tree (as determined by the Huffman table value) and then decoded as a normal block of block floating point quantized mantissas.

B. Bit Reservoir

The Huffman coding gains are tracked and stored in a bit reservoir. The CCRMAC-3 bit reservoir is populated in the encode single channel routine in codec.py. After the optimal bit representation of each block's mantissa vector is calculated, the difference between the theoretical bit budget for the global data rate and the optimal bit allocation is added to the bit reservoir.

Bits are drawn from the reservoir on short blocks (as described in the block switching section) only, as their theoretical bit budget is often far too starved to accurately encode harmonic components of the signal. During bit budget calculation for short blocks, the minimum of the theoretical bit budget and remaining reservoir bits is added to the bit budget. In this way, the bit budget is never more than doubled. This spreads the reservoir over more blocks, as opposed to using all reservoir bits on a single block.

C. Transient Detection

Transient detection is a key component of block switching in a perceptual audio coder. CCRMAC-3 uses a three part transient detection algorithm, which draws from strategies discussed in [1]. Each detection component relies on a comparison between consecutive audio signal blocks. Transients are most easily detected as changes in the high frequency content of a signal. Therefore, prior to analysis, the signal is passed through a 4th order Butterworth highpass filter with cutoff frequency at 0.35*Fs/2.

The first comparison is between the time-domain energy of the first and second half of the current block of data. We calculate the energy of a signal as follows:

$$E(n) = \sum_{m=0}^{N/2-1} [x_h(m)]^2$$

Where $x_h(m)$ is the mth sample of the filtered signal block and E(n) is the energy of the first half of block n. We then take the absolute value of the first half energy subtracted from the second half energy to obtain the amplitude energy difference of the block.

The second and third comparisons are spectral energy comparisons. We use an FFT of the Hanning windowed signal to move to the frequency domain. The spectral energy of block n is calculated as follows:

$$\mathrm{SE(n)}=1/\mathrm{N}\cdot\sum_{\omega=0}^{N-1} |X_l(\omega)|^2 \cdot W(\omega)$$

Where, $\mathrm{W}(\omega) = \omega$, leading to an increased weighting of higher frequency components. For the second comparison, we find the inter-block spectral energy difference SE(n+1)-SE(n). And finally, for the third comparison we find the intra-block spectral energy difference |SE(nFirstHalf)-SE(nSecondHalf)|.

Through extensive testing, we determined thresholds and weightings of the three previous energy measurements, which trigger the detection of a transient. CCRMAC-3 detects transients in the next block of signal data, providing enough time to transition from long blocks to short blocks. This implementation is discussed further in the next section.

D. Block Switching

Block switching is employed to reduce pre-echo noise artifacts which occur when quantization noise gets distributed into the portion of the signal preceding the transient temporally. To reduce the effect, when a transient is detected the system needs to switch to a shorter block length to reduce the likelihood of pre-echo being audible. In CCRMAC-3, a block switching process similar to that used in the AC-2 coder [2] is employed where the transition windows are composed from half the long window and half the short window. The long block length is 2048 samples, the short block length is 256 samples and the transition block lengths are 1152 samples. When a transient is detected, the block prior to the transient is processed with a start transition block and then seven short blocks. If a second transient occurs in the block following the initial transient then processing continues using eight consecutive short blocks. Otherwise, the next block is processed with a single end transition block. The order of processing is implemented so that the amount of data read from the input PCM file at each read is the constant value of 1024 samples. The analysis and synthesis windows used on the blocks are sine windows.

The block switching process is illustrated in figure 2.

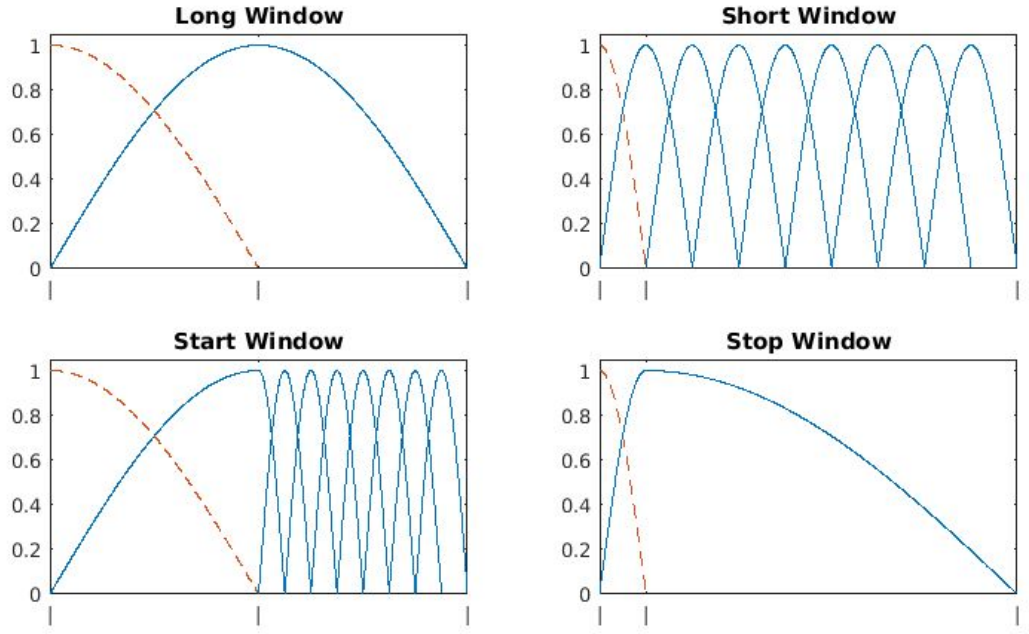

Figure 2. Example long, short and transition blocks for block switching implementation

In order to implement block switching, the following changes were made to the header and block data structures of the PAC file format:

- **nMDCTLines** has been replaced in the header by **nMDCTLinesLong** and **nMDCTLinesShort** which indicate the half block length of long and short blocks.
- **nSFBands** and **nLines** have been removed from the header as they are redundant and can be recalculated by psychoac.py in the decoder
- **blockType**, a three bit field has been added to the data block to indicate the type of block. The values passed are: 0 = long, 1 = start, 2 = short and 3 = stop

IV. RESULTS

To test the quality of the CCRMAC-3 coder against the CCRMAC-2 base coder, we ran an ITU-R BS.1116 listening test on five critical audio samples. We tested the encoded/decoded audio samples at target bit rates of 96kbps and 128 kbps for CCRMAC-3 and 128 kbps for the base coder.

File	Original Size	Encoded Size	Compression Ratio
Castanets	1.6 MB	270 KB	5.92
Glockenspiel	4.6 MB	844 KB	5.45
Harpsichord	2.9 MB	506 KB	5.73
Quartet	4.1 MB	753 KB	5.44
German Speaker	2.9 MB	554 KB	5.23

Figure 3. Compression ratio for CCRMAC-3 target bit rate of 128 kbps (2.9 bps)

Our participant pool consisted of the three team members. The test was administered using the provided listening test graphical python program with Sony MDRZX110 Noise Cancelling Headphones at a constant volume. The listening test results are presented in figure 4 below.

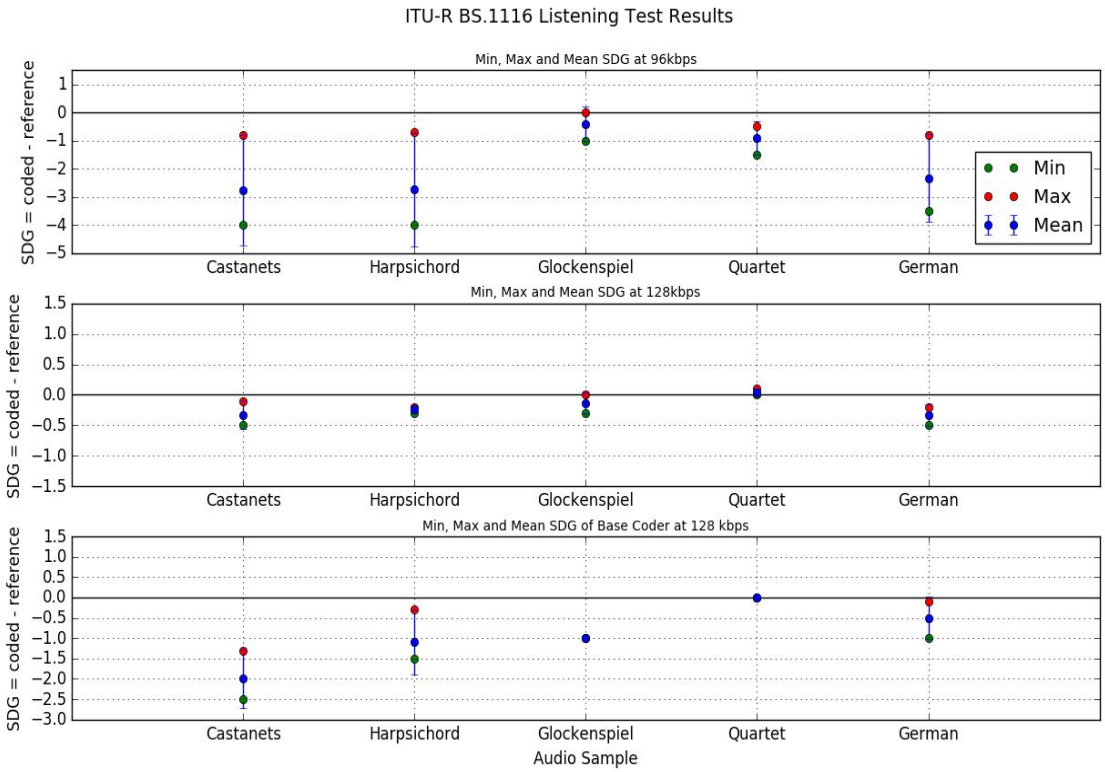

Figure 4. Minimum, maximum and mean (with 95% confidence interval) SDG for several critical audio samples

The CCRMAC-3 perceptual audio coder has an average SDG over all audio samples of -2.33 (minimum -2.767 on castanets, maximum -0.4 on glockenspiel) and -0.33 (minimum -0.33 on castanets , maximum 0.033 on quartet) at 96 kbps and 128 kbps respectively. The base coder has an average SDG of -0.92 (minimum -2 on castanets, maximum 0 on quartet). CCRMAC-3 performed better than the base coder on all critical audio samples at 128 kbps by an overall average of -0.7214. The maximum increase in performance occurred on the castanets audio file (1.67 increase in SDG) and the minimum increase in performance occurred on the quartet audio file (0.033 increase in SDG).

V. CONCLUSION & FUTURE WORK

In this work we demonstrated the CCRMAC-3 perceptual audio coder with implementations of features including entropy (Huffman) coding, bit reservoir, transient detection and block switching. As demonstrated in the results, the CCRMAC-3 perceptual audio coder, at a data rate of 128 kbps, shows significant improvements compared to the base coder, and was able to achieve nearly transparent audio quality in allat mostat most of our testing cases. The results at 96 kbps were less satisfactory and imply that further work is needed to achieve transparent compression at that data rate.

There are a few different aspects of CCRMAC-3 which could be improved upon to reach that target. Firstly, since the encoder reads in a fixed block size from the input data file, this forces a set number of short blocks to be calculated anytime a transient is detected. If the encoder were modified to read in a variable length input, it would be possible to more strategically implement the block switching so as to use the fewest number of short blocks necessary for any particular transient.

The Huffman coding process could be improved with further work on the Huffman tables. This could be done by generating the tables with a larger amount of training data and/or adding additional Huffman tables.

Due to amount of data stored with each data block in the compressed file, the data rate can only be lowered a certain amount before the number of bits allocated for storing mantissas becomes negligibly small or negative. One way to save block data space would be to only pass information such as scale factor bands for bands for which bits have been allocated. Another method to reduce data rates would be to implement spectral band replication which would reduce the amount of data that needs to be transmitted as only a portion of the lower frequency of the spectrum actually gets encoded. A third method would be to reduce the number of spectral bands processed for short blocks so that bits get allocated more efficiently on short blocks.

Stereo coding techniques also were not implemented in CCRMAC-3. Techniques such as M/S stereo coding and intensity stereo coding could be employed to further reduce achievable data rates and reduce coding artifacts in stereo audio recordings.

Finally, the performance of the KBD window was not evaluated in CCRMAC-3 to determine whether it would outperform the sine window. This could also be investigated in the future.

VI. REFERENCES

[1] J. P. Bello, L. Daudet, S. Abdallah, C. Duxbury, M. Davies and M. B. Sandler, “A Tutorial on Onset Detection in Music Signals”, *IEEE Trans. Speech Audio Process*., vol. 13, no. 5, pp. 1035-1047, Sept. 2005

[2] M. Bosi and R. E. Goldberg, *Introduction to Digital Audio Coding Standards*, Springer Science+Business Media, LLC, New York, NY, 1st Edition, 2003