Music 422 Project Report

Doe Hyun Yoon, Dong In Lee

1. Design Considerations

As the aim of this project, we chose to reduce the data rate as low as possible while maintaining good sound quality. So we applied a couple of techniques, but most of our efforts were concentrated on entropy coding, because we believed that it would give the most impressive coding efficiency improvements.

At first, we concentrated our efforts on basic elaborations of our initial coder; local adaptation of bitrate, bit allocation scheme, and MS stereo.

Then, we investigated on various entropy coding technologies including Golomb-Rice coding, Huffman Coding, Arithmetic Coding, and etc. We could have tried advanced techniques such as context adaptive arithmetic coding, but project term was too short to do that. So, we chose Golomb-Rice coding on most of custom designed variable length code generation, since Golomb-Rice code is very flexible and simple to make, and also it is very powerful when the distribution shows roughly geometric distribution.

2. Encoder Description

The overall encoder operation is the same to the basic MDCT based coder.

2.1 Local adaptation of bitrate by variable frame length

In the conventional audio coding, the bit reservoir is designed to support a locally-variable data rate and mitigate possible pre-echo effects [1]. However, handling of bit reservoir needs tedious manipulation of bitstream; some parts of the previous frame shall be stored in the buffer; while decoding of the current frame needs this buffered previous frame data.

Instead of bit reservoir, we achieved the same functionality by the variable frame size. We put the frame size in bytes as a frame header, so that variable sized frame could be decoded easily. If the size of a frame is lower than expected, then the remaining bits can be used for the next frame.

The following pseudo code shows how the local adaptation works with variable data rate demand.

```
BitsPerFrame = bitrate * (N/2) / sample_rate;
maxBitsPerFrame = 2*BitsPerFrame;
minBitsPerFrame = (½)*BitsPerFrame;
bitsdiff = 0;

for all frames encoding
      target_frame_size = BitsPerFrame + bitsdiff;
      if( target_frame_size > maxBitsPerFrame )
        target_frame_size = maxBitsPerFrame
      if( target_frame_size < minBitsPerFrame )
        target_frame_size = minBitsPerFrame
```

```
encode a frame, where the target frame size is target_frame_size;
bitsDiff = target_frame_size - encoded bits;
```

Note that target_frame_size is limited to maxBitsPerFrame and minBitsPerFrame, which are the twice and half the nominal frame size, respectively.

Figure 1 shows the effectiveness of this technique. It shows the actual generated bits (red line), desired target bits (blue line) of each frame for the case of 64kbps harpsichord. It should be noted that the generated bits varies frame by frame, while the average frame size is roughly the same as that of constant bitrate target. Also, we can encode the sudden re-start of music after a few seconds of silence with the nearly the twice the nominal target frame size.

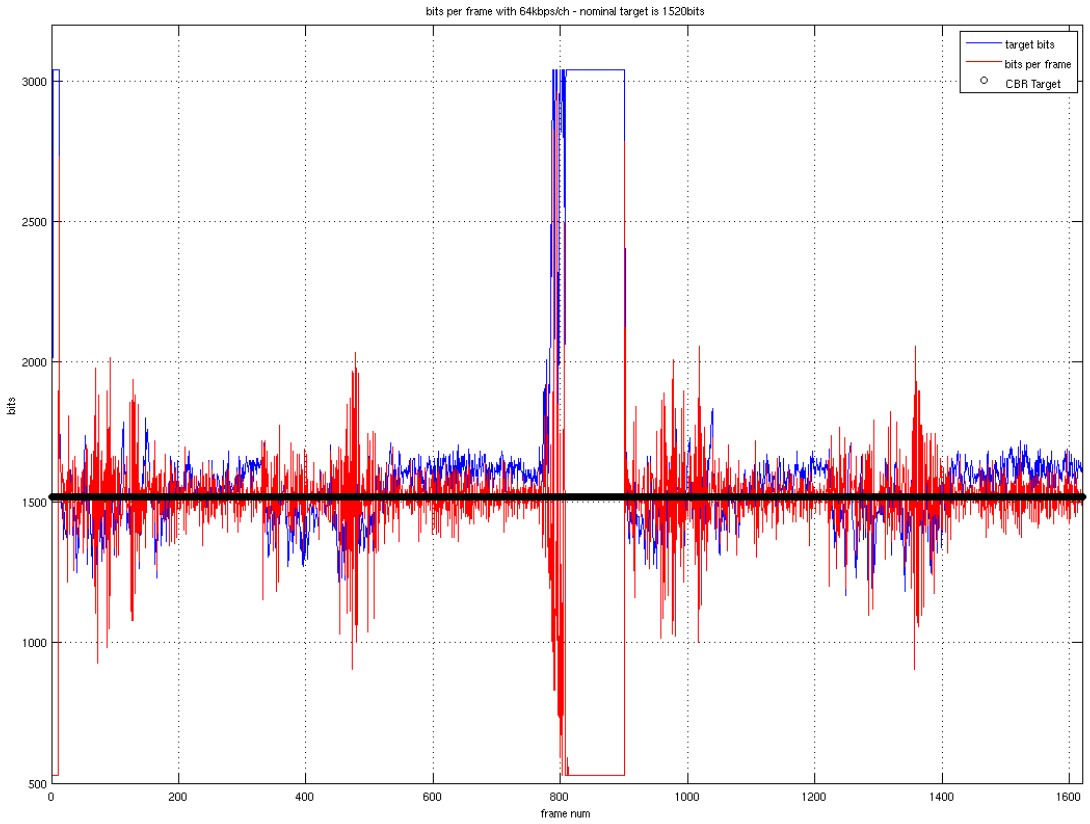

Figure 1. Results of local adaptation of bitrate for 64kbps harpsichord

2.2 Bit allocation

We had two choices of bit allocation schemes; optimal bit allocation and water-filling. While the optimal one is claimed to be theoretically optimal, it may have some negative effects on the practical application [1]; negative number of bits might be allocated to some subbands, 1 bits could be assigned to some subbands (but we can't assign 1bits in our block floating point quantizer based on midtread quantization scheme).

So, we've implemented “water-filling” algorithm to achieve both the practical and optimal bit allocation. But, the problem regarding the water-filling was that it doesn't allocated any more bits once

all the subbands have SMRs below 0dB. This may causes some problem especially with regards to variable length coding, where this bit allocation doesn't work correctly. Thus, we may need to run this bit allocation routine with far higher target bitrate than the nominal target (refer to the bitrate control section). Our empirical experiments with harpsichord showed that water-filling generated the same size of encoded file when the target bitrate is beyond 110kbps.
Our way to cope with this problem is simply not to stop bit allocation on the SMR 0dB, and continue the bit assigning procedure of water-filling algorithm until the bit pool drains to 0bits no matter what SMR values each subband has. In fact, as long as there are remaining bits, it's better to allocate more bits and lower SMR always ensures a better audio quality.

2.3 MS Stereo coding
We tried to implement MS stereo coding based on [2]. Thus, the M/S coding is applied when the average of differences of SMRs of left and right channels over all 25 scale factors are below 2dB.

LR to MS conversion is done as follows;

```
M = (L+R)/2;
S = (L-R)/2;
```

MS to LR conversion in decoder side is as follows;

```
L = M+S;
R = M-S;
```

This LR to MS conversion is applied on MDCT coefficients, and MS to LR conversion would take place in decoder again on decoded MDCT coefficients, so there's no problem with the frame adaptive LR/MS coding.

However, the detailed bit allocation proposed in the paper needs to know the masking curves of Mid / Side channels, while it requires Mid/Side channel basic masking threshold and masking level difference factor described in [3, 4]. We though that it's too complicated procedure. Hence, instead of doing the exact procedure proposed in the original paper, we used a simple bit reallocation strategy here.

If M/S stereo is applied to a frame, we can expect that most of signal energies are compacted into Mid channel, and Side channel would have less energy. Since we already know the optimal (at least, it is claimed to be optimal) bit allocation of left / right channels, the sum of left bits and right bits are reallocated into Mid channel and Side channel for each scale factor bands. The following pseudo code shows detailed operation.

```
For each scale factor band s,
      bitsR = rm of right channel of a scale factor band s;
      bitsL = rm of left channel of a scale factor band s;
      bitsSum = bitsR + bitsL;
      energyM = sum of signal energy of Mid channel of a scale factor band s;
      energyS = sum of signal energy of Side channel of a scale factor band s;
```

```
bitsM = bitsSum * energyM / (energyM + energyS);
if( bitsM > 15 ) bitsM = 15;
else if( bitsM  == 1 ) bitsM = 0;
bitsS = Sum – bitsM;
```

Through this re-allocation procedure, we can assign mantissa bits of Mid / Side channels from the bit allocation results of left / right channel, while the complex masking threshold calculation of Mid / Side channels could be avoided.

2.4 Entropy coding

There are Rs, Rm for each scale factor band, and mantissa values for each 512 MDCT coefficients. However, we didn't care about how to code with Rs and Rm bits since they are only 200 bits in total and most of bits would be consumed by 512 MDCT coefficients. Hence, the variable length coding is applied to MDCT coefficients only.

Each mantissa could be in the range between -(2^(Rm-1)-1) and +(2^(Rm-1)-1). With that in mind, our coding strategy is split mantissa into sign and magnitude except Rm being equal to 2.

Once the sign values (0 or 1) are aggregated to one signal data stream, we could easily apply the run-length coding followed by Golomb-Rice coding. Moreover, it becomes easier to make codebooks with absolute values, since the number of possible cases in absolute value is half the cases of the signed value (e.g. for Rm=8, signed value could be between -127 ~ to +127, total 255 cases, while the absolute value would be 0 to +127, total 128 cases)

2.4.1 Sign coding - run-length coding followed by Golomb-Rice coding

Since there are 512 MDCT coefficients, we would have 1024 sign values considering stereo. However, signs of coefficients belonged to subbands having Rm being equal to 0 or 2 are omitted; subbands of Rm 0 would not be coded, while subbands of Rm 2 would be coded with variable length coding of signed values.

This stream of 0 and 1 are first run-length coded, then the run-length values are encoded with Golom-Rice coding with M being equal to 2 [5].

2.4.2 Analysis on the signal distribution of absolute value of mantissa

Figure 2 shows that the actual absolute value of mantissa distribution classified based on Rm values. It is clear that the distribution of low Rm value was quite similar to geometric distribution, but it isn't like that with high Rm value. Especially, when Rm is 8 or higher, the distribution becomes quite smeared over the entire range.

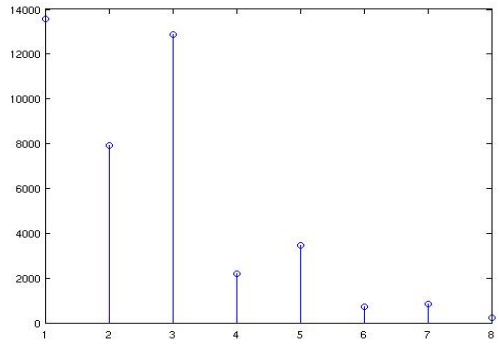

Fig 2.a) Rm=4

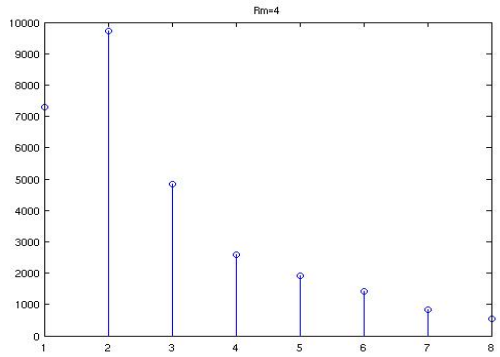

Fig 2.b) Rm=4

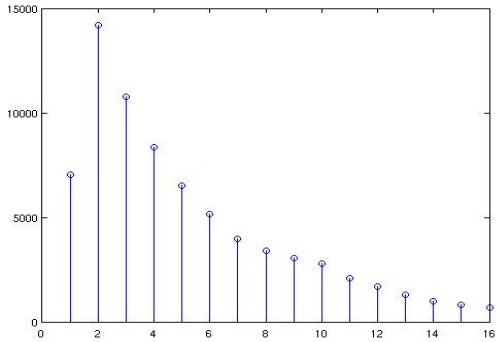

Fig 2.c) Rm=5

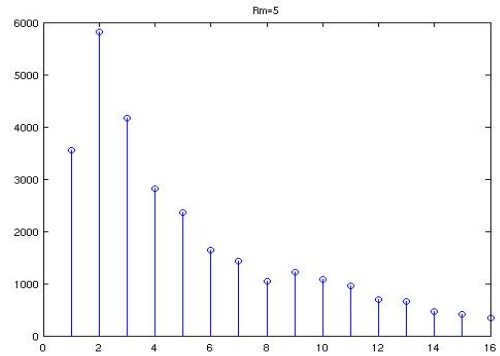

Fig 2.d) Rm=5

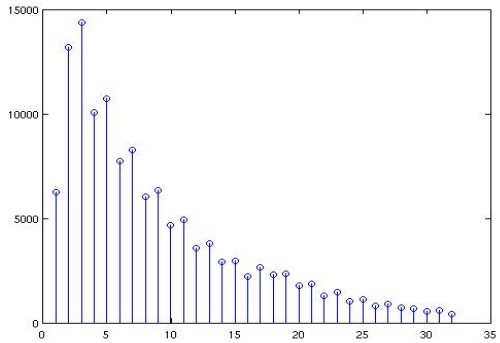

Fig 2.e) Rm=6

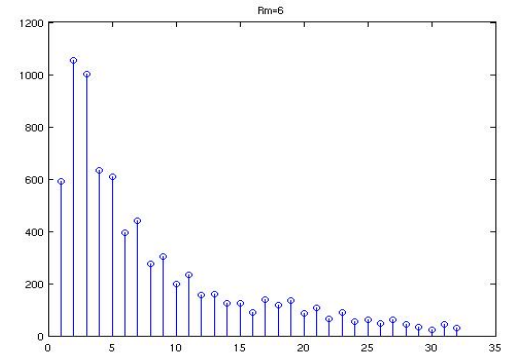

Fig 2.f) Rm=6

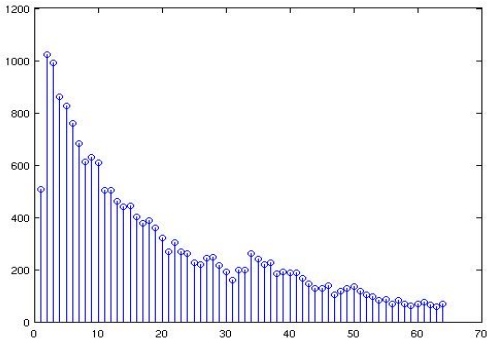

Fig 2.g) Rm=7

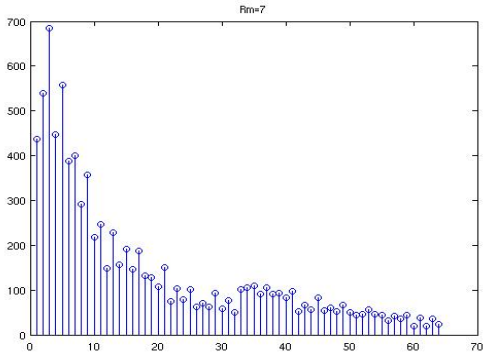

Fig 2.h) Rm=7

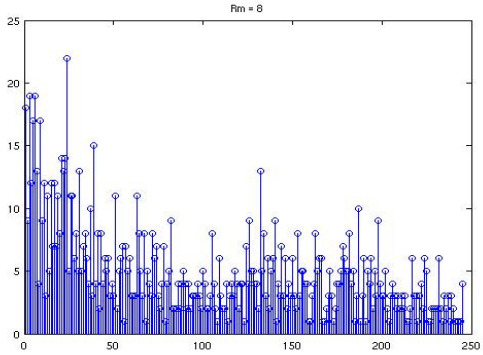

Fig 2.i) Rm=8

Figure 2. absolute value of mantissa distribution based on Rm

2.4.3 Code book generation

Initially we thought that each coefficient would be variable length coded, but studying on coding standards such as MP3 [6] and AAC [7] revealed that most advanced coding standards code together a couple of MDCT coefficients to form one variable length code word. Hence, more detailed investigations were carried out on AAC coding standard [7] and open source AAC software FAAC [8]. As a result, we decided to use variable length code for 2 or 4 tuples of data.

In fact, one AAC code book matches exactly to our usage on Rm 2 case, so we borrowed that code book. For the code books of Rm 3,4,5,6, and 7, they are generated as follows;

On each entry, we have a codeword, codewordlength, and n-tuple data, where each data ranges 0 to (2^(Rm-1)-1).

All the entries are sorted based on the sum of energies of n tuple data (e.g., 4 tuples 0, 2, 3, 1 would have the sum of energy 0^2 + 2^2 + 3^2 + 1^2 = 14)

codeword and codewordlength are generated with Golomb-Rice coding [5, 9] with appropriate M values

(we tried a couple of M values, then selected empriacally).

2.4.4 Rm = 2 mantissa coding - 4 tuples of signed values

Each MDCT coefficients of Rm 2 could be one of -1, 0, and +1, so the codebook of 4 tuples is 3^4=81 entries. This codebook is used to code four consecutive MDCT coefficients resulting one variable length codeword.

2.4.5 Rm = 3 - 4 tuples of unsigned values

Except the absolute value is used, the coding of Rm 3 coefficients is identical to Rm 2 coefficients. Since the absolute value ranges 0 to 3, the codebook size is 81 entries, and likewise the previous case, four consecutive coefficients are encoded to one variable length codeword.

2.4.6 Rm = 4,5,6,7 - 2 tuples of unsigned values

Since the number of possible cases increases exponentially as Rm increases, it was impossible to utilizes 4 tuple codeword. So 2 tuple codeword is applied Rm 4,5,6, and 7, the two consecutive MDCT coefficients are encoded to one codeword.

2.4.7 Rm > 7 - fixed length coding of unsigned values

For Rm greater than 7, it was nearly impossible to apply the full code word even for 2 tuple code book. Maybe we could apply escape code followed by a couple of fixed length code to cope with large signals while the code book size is kept to be small.

However, initial trials with escape code showed some negative effects; the bitrate increased due to so many cases of larger signals coded with escape code. We suspected that it is due to the fact that block floating point quantizer normalizes the given coefficient, but not are sure about that.

2.4.8 Evaluation of variable length coding performance

Figure 3 given below shows the effectiveness of our entropy coding. This is how much bitrate is reduced with the application of variable length coding including sign run-length coidng. Note that for this experiment, tools like local adaptation of bitrate or iterative bitrate control described in the following section are turned off for fair comparison.

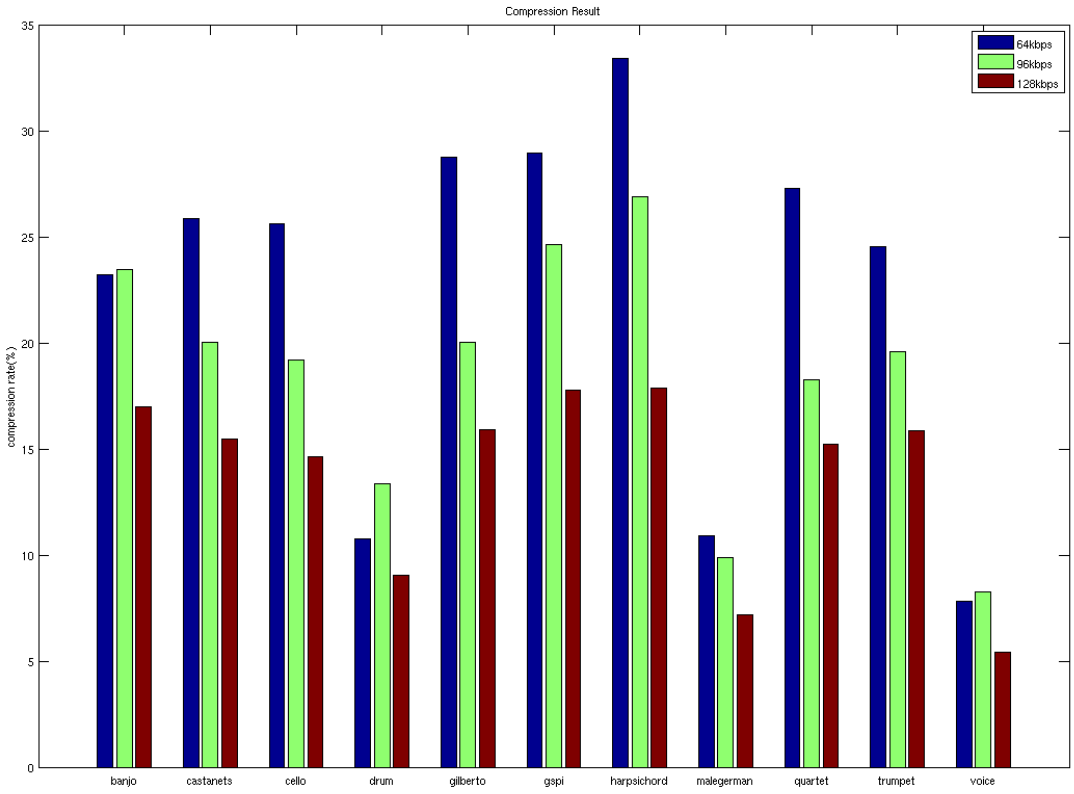

Figure 3. Bitrate reduction with variable length coding

2.5 bitrate control

All the assumptions we made on the bit allocation was based on fixed length coding, that is 6.02dB SNR per each bit. However,with the variable length coding, this rule doesn't work at all. There could be various ways to meet target bitrate with variable length coding, but we wanted to keep it simple.

Hence, we applied iterative control of bitrate to meet the target bitrate with variable length coding The detailed algorithm is as follows.

First, the target bitrate is set as an argument, and huffman compression is executed. After we get the compresed file, we can now calculate the difference between the original target bitrate and result bitrate from the first iteration. Next, if the result bitrate is smaller than the original target bitrate, we should increase the target bitrate at the next iteration. Conversely, if the result bitrate is larger than the original target bitrate, we should decrease the target bitrate at the next iteration. To adjust bitrate, we simply set the difference between current bitrate and the original target bitrate as the adjustment. Finally, if the difference is smaller than 0.5, the iteration ends. In addition, the iteration is executed ten times at maximum. The following pseudo code shows detailed operation.

```
If ( result_bitrate are enough close to target bitrate  ) {
      We are done
 } else if( result_bitrate is smaller that target bitrate ) {
      Calculate the difference between result bitrate and target bitrate
      Add the difference to current bitrate for the next iteration
      Restart the whole encoding procedure again
 } else{
      Calculate the difference between result bitrate and target bitrate
      Subtract the difference to current bitrate for the next iteration
      Restart the whole encoding procedure again
 }
```

We defined “effective bitrate” as the bitrate of the same sound quality without any variable length coding. This would be a metric of how much our variable length coding increases the sound quality given the bitrate constraints. Note that for this experiment, local adaptation of bitrate is turned off for fair comparison.

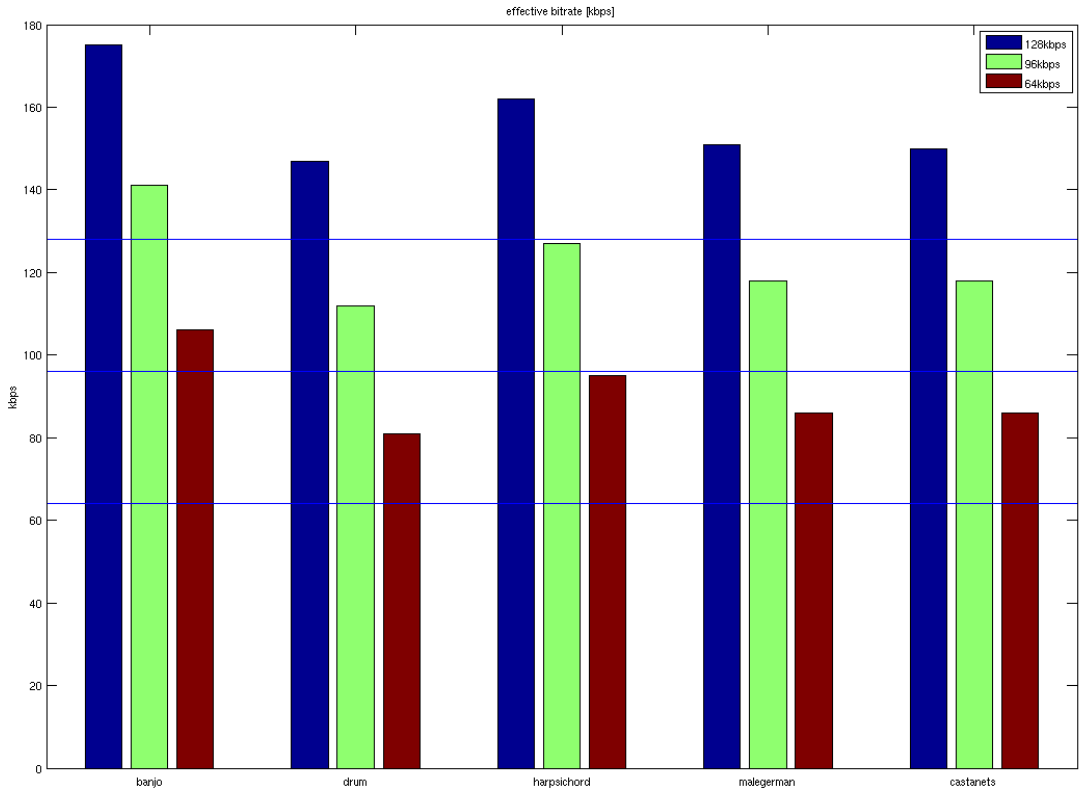

Figure 4. effective bitrate

For example, 64kbps banjo coded with variable length coding is about the same sound quality of 105kbps coded without variable length coding. Most of the cases, the effective bitrates are 20~40kbps higher than the actual bitrate.

3. Bitstream Format

3.1 Top level

```
FileHeader();
while( !EOF ) {
    Frame();
}
```

3.2 File Header

```
FileHeader() {
    channels;           // unsigned 8bits, 1 or 2
    sample_rate;        // unsigned 32bits, e.g., 44100
    total_samples;      // unsigned 32bits,   total number of samples in each
channel
}
```

3.3 Frame Level

```
Frame() {
    FrameHeader();
    ScaleFactorInfo();
    Sign();
    Mantissa();
}
```

3.4 Frame Header

```
FrameHeader() {
    bytes_in_frame;     // unsigned 16bits, number of audio data in this frame
                        // it doesn't include FrameHeader data
    if( channels == 2 )
        ms_stereo;      // unsigned 8bits, 0 for L/R, 1 for M/S
}
```

3.5 Scale Factor Information

```
ScaleFactorInfo() {
    for(c=0;c<channels;c++) {
        for(z=0;z<25;z++) {
            rm[c][z];         // unsigned 4bits, Rm
            if( rm[c][z] != 0 )
                rs[c][z];     // unsigned 4bits, Rs
            else
                rs[c][z] = 0;
        }
    }
}
```

3.6 Audio Data – mantissa of 512 MDCT coefficients

```
Sign() {
    Golomb-Rice encoded Run-length values,
    signs are decoded for the scale factor bands whose Rm > 2
}

Mantissa() {
    For each channel, each subband,
    if( rm == 0 ) mantissa = 0;
    else if( rm == 2) decode 4 tuples of signed values
    else if( rm == 3) decode 4 tuples of unsigned values
    else if( rm >4 and rm <8) decode 2 tuples of unsigned values
    else decode fixed length code
}
```

4. Evaluation

First, we want to discuss the objective improvements. As we described in the encoder description, variable length coding reduces the overall bitrate approximately 15 ~ 25%. With the proper coder control, we achieved “effective bitrate” substantially higher than the actual bitrate. So, this simple objective metric tells us that our coder has about 150kbps/ch quality without variable length coding at 128kbps/ch.

Although the overall quality assessment should be carried out by listening test, we didn't perform any formal listening test. Instead, we asked a coupled of our friends to hear and evaluate our coder at 128kbps/ch and 96kbps/ch, and their responses were as follows;

- With blind test, one couldn't tell the difference between the original orchestral sample and 96kbps/ch encoded file
- Some trained listener noticed some quantization with castanets 128kbps/ch, but he said that the artifacts were very little
- Many people mentioned that the quality of 128kbps/ch was nice, and it was quite good even with 96kbps/ch

Note that this was just an unofficial test with a few listeners using headphones in CCRMA machines.

5. Conclusions and Future Works

We have achieved our goal, a decent audio coder at the bitrate below 128kbps/ch. The main techniques we tried were MS stereo coding and variable length coding in conjunction with proper control of coder. Although the efficiency of the basic MDCT based coder has been improved significantly, there could be various ways of enhancing further.

One thing we planned but didn't complete yet is applying multiple VLC codebooks for each frame; each codebook would be generated by Golomb-Rice coding with different M factors, then the table selection could be made by either explicitly (adding the syntax information which codebook shall be used) or implicitly (the selection of code book is done based on predetermined rule, doesn't need additional syntax information).

If we could have more time, we'd like to try in-depth elaborations on entropy coding; for example recent video / image compression standards utilizes context adaptive arithmetic coding, so we believe that the same concept could be applied to audio coding.

Last thing we want to mention is that audio coding could also take advantages of long term temporal prediction; both the encoder / decoder have the same decoded waveform buffer, and audio data could be predicted from that buffer with the proper displacement vector. This may causes some difficulties in random accessing or transmission error. However, as long as this long term temporal prediction improves coding efficiency, it would be worth while to do.

References:

[1] Marina Bosi, Richard E. Goldberg, Introduction to Digital Audio Coding and Standards, Kluwer Academic Publishers, 2003.

[2] J. D. Johnston and A. J. Ferreira, “Sum-Difference Stereo Transform Coding,” Proc. of International Conference on Acoustics, Speech, and Signal Processing, vol. 2, pp. 569-572, San Francisco, USA, March, 1992.
[3] J. D. Johnston, “Transform Coding of Audio Signals Using Perceptual Noise Criteria,” IEEE Journal on Selected Areas in Communication, Feb. 1998.
[4] K. Brandenburg and J. D. Johnston, “Second Generation Perceptual Audio Coding: The Hybrid Coder,” AES 89th Convention, 1990.
[5] Stanford University, ee398 class – entropy and lossless coding http://www.stanford.edu/class/ee398/handouts/lectures/01-EntropyLosslessCoding.pdf
[6] ISO/IEC 11172-3, Information Technology, “Coding of moving pictures and associated audio for digital storage media at up to about 1.5Mbit/s, Part3: Audio”, 1993.
[7] ISO/IEC 13818-7, “Information Technology – Generic Coding of Moving Pictures and Associated Audio, Part 7: Advanced Audio Coding”, 1997.
[8] Free Advanced Audio Coder project, http://sourceforge.net/projects/faac/
[9] H. S. Malvar, “Adaptive Run-Length/Golomb-Rice Encoding of Quantized Generalized Gaussian Sources with Unknown Statistics,” Proc. of Data Compression Conference, Issue 28-30, pp. 23-32, March, 2006.