Rego Sen, Wai Kit Leung, Jeremy Liu
EE 367C / Music 422
Final Project
March 12, 2004

Compression-Optimizing Window-Based Encoding Logic Layer-1

Specifications and Performance Review

TABLE OF CONTENTS:

I. INTRODUCTION

The aim of our final project for EE 367C / Music 422 is to implement a "good-sounding" audio codec at rates of 128 kb/s per channel or less, we will use an MDCT based algorithm to quantize and encode the data. We will use FFT based analysis to generate a psychoacoustic model and perceptually optimally allocate bits for the compressed data. Here is the outline of our proposed stages of progress (note that we didn't expect to complete all the stages in the time alloted):

Stage 1 : Implement basic perceptual audio coder using sine windows, two-slope spreading functions and simple 'addition' of masks (maximum mask value). Incorporate bit packing.

Stage 2 : Tuning perceptual model. Add options to change addition of masks and use different spreading functions. Maybe look at noise vs tone maskers.

Stage 3 : Add Huffman Coding to further compress bit stream.

Stage 4: Additional features: noise reduction algorithms
Encoding Recognition (determine whether the signal has been previously encoded by passing only masked sub-bands and checking if the result is noise)

Stage 5: Add Hanning windows and Kaiser-Bessel Windows

We successfully completed stages 1 and 2.

II. USER INTERFACE

The encoder and decoders are executable from a Linux terminal. The syntax for running each is as follows:

```
SYNTAX: encode inputfile outputfile [-s spreading function] [-a alpha] [-n blocksize] [-r data rate]

        inputfile:      name and path of original file
        outputfile:     name and path of encoded file to be (over)written
        spreading functions:
                0: two-slope  <-- DEFAULT SPREADING FUNCTION
                1: Model 1
                2: Model 2
                3: Schroeder
                4: Terhardt
        default alpha = Infinity (maximum of each mask)
        default blocksize = 1024
        default data rate = 256 kb/s/ch
```

```
SYNTAX: decode inputfile outputfile

        inputfile:      name and path of encoded file
        outputfile:     name and path of output file to be (over)written
```

The input and output files must be specified in order to run either program. Otherwise, the programs will display their appropriate syntax as shown above.

III. KEY FEATURES

A. ADDITION OF MASKS

In creating the overall mask used in our perceptual model, we needed some way to sum the spreading functions generated by different maskers. Using the equation found on page 192 of Bosi and Goldberg's Digital Audio Coding and Standards[1], we can sum the masking curves produced by multiple maskers in various ways by varying alpha between 1 and infinity. When alpha is 1, the masking curves are summed in intensity. When alpha is infinity, the largest value of a mask at any point is taken. We provided a command line argument that allows the user to change the alpha used in encoding. The default value is infinity.

B. SPREADING FUNCTIONS

We have looked into different spreading functions and compared the effect of masking from using different spreading functions. We also looked into both tone masking and noise masking. In particular, the following spreading functions (analytical expressions for each can be obtained from Chapter 7 of DACS) were implemented and investigated:

i. Two-slope spreading function
ii. Schroeder spreading function
iii. Model 2 spreading function
iv. Model 1 spreading function
v. Terhardt spreading function

We investigated the masking effect of different spreading functions using the following musical excerpts:

a. A synthesized sinusoidal signal, identical to the one using in assignment 4 for the generation of the masking curve
b. An excerpt of oboe and harpsichord, from the opening of the *Sonata in E minor* by Francesco Geminiani. The oboe plays a B4 while the harpsichord plays an E minor chord
c. An excerpt of an orchestral fortissimo, featuring the Leningrad Philharmonic Orchestra performing the opening of the 4th movement (*Adagio*) of Dmitri Shostakovitch's *Symphony No. 8*. The excerpt features the full orchestra playing in fortissimo, with a prominent cymbal roll and snare drum roll

a. The masking curves obtained from the synthesized sinusoidal feature six distinct peaks, not surprising as the masking tone has six distinct (i.e. well-resolved) peaks in the frequency spectrum. The Schroeder spreading function provides the most masking, as well as a smooth peak.

b. The masking curves obtained from the oboe and harpsichord excerpt have peaks in several frequencies. The tone of the oboe is rich in harmonics -- in fact in oboe's fundamental register (which includes the B4 played here) the first overtone is always stronger than the fundamental, and the next few overtones are also prominent. Coupled with the overtones from the harpsichord, the masking curve has peaks at B4 as well as at B4's harmonics (and harmonics of the harpsichord tone as well).

c. The orchestral excerpt is dominated by the tone of the brass (the trumpets being especially prominent -- a feature of the Leningrad Philharmonic under Evgeny Mravinsky) and as a result there are a number of well-resolved peaks in the frequency spectrum, as evidenced in the numerous and distinct peaks in the masking curve.

The masking curves for these cases are included in the appendix. All three graphs were generated using a two-slope spreading function, with alpha set at infinity and block size of 512.

C. TONE VS. NOISE MASKING

---

1. $$I_N = \left(\sum_{n=0}^{N-1} I_n^{\alpha} \right)^{\frac{1}{\alpha}} \qquad 1 \le \alpha \le \infty$$

A tone will mask sound that is up to 15 dB below its peak, while noise will mask sound that is up to 6 dB below its peak. We took advantage of this by boosting the default spreading functions by 9 dB when detecting that the masker is noisy. There is no universal formula for discerning narrowband noise from a tone, so today's state-of-the-art codecs employ different methods to achieve this goal. For example, MPEG 1 treats only the highest peak in a transform window as a tone masker, and everything else as a noise masker. We feel that this approach was suboptimal, and instead we have our own formula (used in mask.c):

A peak is treated as a noise masker if its distance from the previous peak is less than 5 times the natural log of its frequency.

Taking the natural log of the frequency allows greater bandwidth for high-frequency noise. At first we considered basing the function on the bark scale, but that allowed too much low-frequency noise to be treated as tone maskers. Also, taking the base 10 log was either too conservative for high-frequencies or too liberal with low-frequencies (depending on the coefficient). Using a coefficient of 5 was sub-optimal for frequencies below 100 Hz, but that region allocates the smallest number of frequency bins compared to other sub-bands. Since the coefficient of 5 works very well for the higher frequencies, we found it to be the optimal choice.

IV. EVALUATION OF CODER

A. SAMPLES USED FOR TEST CASES

We used a sum of sine tones as a basic test case. We also chose several samples from the EBU SQAM disc that can be found at:
http://www.tnt.uni-hannover.de/project/mpeg/audio/sqam/
The first of those samples was a sample of female speech. The second was a clip containing classical guitar played along with castanets. The last sample was that of a glockenspiel.
As previously noted in this report, we also used musical excerpts from our own CD collection. The excerpts used included the opening of the 2nd movement *Allegro* of the *Sonata in E minor* for oboe and continuo by Francesco Geminiani, performed by oboist Jacques Vandeville and harpsichordist William Christie, and the opening of the 4th movement *Adagio* of the *Symphony No. 8* by Dmitri Shostakovitch, performed by the Leningrad Philharmonic Orchestra under Evegeny Mravinsky.

B. ENCODER PARAMETERS USED FOR TEST CASES

We encoded the samples using a block size of 1024 and data rates of 128 kbps and 256 kbps. For the majority of the test we left spreading functions and the addition of masks value alpha to their defaults: the two-slope spreading function and alpha = infinity. We then used different spreading functions and varied the value of alpha in order to study their effects on the perceptual quality of the output files.

C. RESULTS

C.1. Effect of different spreading functions

Different spreading functions indeed gave different-sounding samples, although the differences were minimal, often to the point of initial imperceptibility; however, repeated and careful listening reveals that each spreading has its own character, which is dependent on the particular musical excerpt used.

The sound files can be found at
http://www-ccrma.stanford.edu/~wkleung/422/

C.2. Effect of varying alpha

Different values for alpha were used and there were differences among the coded files as well. For the Shostakovitch clip, with a two-slope spreading function, alpha = 10000 seemed to produce less-pronounced artifacts than alpha = infinity. alpha = 1 and alpha = 10 produced maximal masking -- the output file in each case was total silence for 20 odd seconds!!!

C.3. Effect of varying the bit rate

Sound files for the samples can be found at:
http://www-ccrma.stanford.edu/~jliu/422/
At 128 kbps, we achieved compression ratios of about 5.5:1. For the sine tones, we felt the encoder was transparent for both data rates. Most of the other samples seemed transparent at the 256 kbps data rate, but artifacts were audible at 128 kbps. In particular, the transient attacks of notes seemed to be quite noisy. For example, the castanets in the guitar example had audible pre-echo and sounded 'crunchy', while the harpsichord in the harpsichord and oboe example sounded like noise had been added to the attacks.

V. CONCLUSION

Our goal was to create a "good-sounding" audio codec at rates of 128 kbps or less. Acknowledging that "good-sounding" is a fairly

subjective criterion at best, we feel we did not quite reach that goal, although we came close. Slightly annoying artifacts were still present at rates of 128 kbps. There are some possible things we could have done to improve the situation. One would have been to implement Hoffman coding, to give us more bits to use. Implementing block-switching would have helped get rid of the pre-echo. Also, replacing the sine window with a Hann window in the side-FFT and a Kaiser-Bessel derived window in the MDCT might have increased the 'peakiness' of our spectra and helped with our bit allocation.

We managed to get through two stages of our planned implementation, which was good for our two-week time frame. The addition of variable spreading functions, user-defined alpha values for the addition of masks and distinguishing noise maskers were all extra features.