IC Project 1

November 2023

universidade
de aveiro

Contents

1 Introduction

The field of Information and Coding offers vast possibilities in audio processing and data handling. The lab work aims to explore these aspects through a series of programming tasks, each building upon the functionality provided by the C++ language and the libsndfile library. The tasks include audio analysis, error measurement, data compression, effect generation, and bitstream manipulation. The culmination of this lab work is the development of a lossy audio codec that leverages DCT for mono audio files. The effectiveness of these implementations is measured in terms of audio fidelity, compression ratio, and processing efficiency.

2 Initial Setup

The initial setup involved the installation of sndfile-example.tar.gz, which is crucial for reading and writing sound file formats. The libsndfile library, paired with its C++ wrapper, provided the necessary groundwork for audio file manipulation.

3 Part 1

3.1 WAVHist

The WAVHist class was modified to calculate histograms for the MID and SIDE channels. These histograms were configured to support coarser bins, combining multiple sample values into single histogram bins for a more compact representation.
In order to visualize the values, a graphic interface was built using OpenCV and fed with the data read and processed from the audio files.

3.2 WAVCmp

The WAVCmp program was built with the template of the base program, adding functions to calculate the mean square and infinite norms and also the signal-to-noise ratio.

3.3 WAVQuant

The WAVQuant was built based on the WAVHist, only adding a method to perform the uniform scalar quantization based on a signal described on time and the number of quantization levels to be used on the quantization process.

3.4 WAVEffects

The WAVEffects class was also built based on the WAVHist class, in order to have a robust method to read and write the signals. It acts as a template in

which it is possible to implement the application of sound effects in the signals processed. Two example functions were implemented to demonstrate the functionality of WAVEffects: The applyEcho function is designed to simulate the acoustic phenomenon of an echo in a given set of audio samples. By taking into account the sample rate of the audio, it calculates the precise number of samples that correspond to the desired echo delay in seconds. Once calculated, it iterates through the audio samples, starting from the point where the echo should be first heard. For each sample thereafter, it adds a fraction of an earlier sample's value, scaled by the decay rate, to the current sample. This creates a repeating, attenuating reflection of the sound that mimics the effect of an echo bouncing off surfaces in a physical space, with the decay parameter controlling the rate at which the echo fades away.
The applyAmplitudeModulation function manipulates the amplitude of an audio signal to create a vibrato-like effect known as amplitude modulation. It operates by oscillating the volume of the audio at a specific modulation frequency, effectively varying the amplitude of the waveform. As it processes each sample, it applies a sine wave (calculated based on the sample index, the sample rate, and the modulation frequency) that alternately amplifies and attenuates the signal. The depth of the modulation determines the intensity of this effect, with a deeper modulation causing more dramatic fluctuations in volume. The result is a dynamic and rhythmic change to the original audio that adds textural richness and can be used creatively in sound design and music production.

4 Part 2

4.1 BitStream Base Class

The provided C++ program demonstrates the functionality of a custom BitStream class, designed for bit-level manipulation within a file. The program checks for the proper number of command-line arguments before proceeding to manipulate bits within the file specified by the first argument. It utilizes various methods from the BitStream class to showcase fundamental operations such as reading individual bits, modifying a particular bit's value, and writing a sequence of bits back into the stream.

The BitStream class appears to encapsulate operations that allow direct access to specific bits within a file stream, offering methods to both read from and write to the file at the binary level. Notably, the program exhibits the alteration of a single bit within the bitstream, highlighting the class's ability to handle precise bit manipulation. This is essential for tasks that require detailed control over data, such as error correction or encoding processes.

4.2 Bitstream Encoder/Decoder

Furthermore, the program showcases the BitStream class's encoding and decoding capabilities. It performs these operations by opening the target file and a file for the encoded output, respectively. The encoding process converts textual representations of binary data into actual binary format, while the decoding process reverses this operation. Such functionality could be crucial in applications where compact binary representation and the subsequent reconstruction of data are necessary, as is common in data compression and transmission algorithms.