Audio effects

Ferdinand Kulman

December 2024

1 Introduction

Audio effects play a crucial role in music production, enhancing and shaping the sound to create unique textures and depth. These effects are widely used in recording studios and live performances to manipulate audio signals for artistic purposes. My task was to implement a selection of common audio effects in MATLAB, including reverb, distortion, chorus, delay, and one freely chosen effect. Each effect was created from scratch to deepen the understanding of their underlying principles and signal processing techniques.In the following sections, I will document the implementation of these effects, detailing the algorithms used, the mathematical basis behind them, and examples of their application to audio signals.

2 Reverberation Effect Implementation

2.1 Function Overview

The `customReverberator` function implements a reverberation effect for audio signals using a network of comb and all-pass filters. Reverberation simulates the persistence of sound as it reflects and decays in an acoustic environment.

```
function reverbSignal =
customReverberator(inputSignal, Fs, reverbTime, wetDryMix)
```

2.2 Input Parameters

The main function accepts the following input parameters:

- `inputSignal` (array): Input audio signal.
- `Fs` (scalar): The sampling frequency of the audio signal (in Hz).
- `reverbTime` (scalar): The reverberation time in seconds, which determines the decay of reflections.
- `wetDryMix` (scalar): The mix ratio between the processed (wet) and original (dry) signal. Valid range: $[0, 1]$.

2.3 Output

The function returns the following output:

- `reverbSignal` (array): The audio signal with the applied reverberation effect.

2.4 Supporting Functions

2.4.1 Comb Filter (`combFilter`)

The `combFilter` function implements a simple feedback delay network used for simulating discrete echoes.

- `input` (array): The input audio signal.
- `delaySamples` (scalar): The delay length in samples.
- `gain` (scalar): The feedback gain.
- `output` (array): The filtered audio signal.

2.4.2 All-Pass Filter (`allPassFilter`)

The `allPassFilter` function implements a phase-shifting filter used to smooth the sound of the comb filter network.

- `input` (array): The input audio signal.
- `delaySamples` (scalar): The delay length in samples.
- `gain` (scalar): The feedback gain.
- `output` (array): The filtered audio signal.

2.5 Algorithm Description

The reverberation effect is implemented as follows:

1. Normalize the input signal to prevent clipping.
2. Define an array of delay times and gains for the comb filters. The gains are calculated based on the specified reverberation time using the exponential decay formula:

$$\text{gain}_i = \exp\left(-3 \cdot \text{delay}_i / \text{reverbTime}\right).$$

3. Process the input signal through multiple comb filters, summing the outputs.
4. Pass the summed output through a series of all-pass filters to smooth and diffuse the sound.

5. Mix the processed (wet) and original (dry) signals based on the `wetDryMix` parameter.

6. Normalize the final signal to ensure that it remains within the valid range.

2.6 Mathematical Representation

The output signal is computed as:

$$\text{reverbSignal} = \text{wetDryMix} \cdot \text{processedSignal} + (1 - \text{wetDryMix}) \cdot \text{inputSignal}.$$

3 Chorus Effect Implementation

3.1 Function Overview

The `customChorus` function implements a chorus effect for audio signals. The chorus effect simulates multiple slightly detuned versions of the input signal being played simultaneously, creating a rich and expansive sound.

```
function chorusSignal =
customChorus(inputSignal, Fs, depth, rate, wetDryMix)
```

3.2 Input Parameters

The function accepts the following input parameters:

- `inputSignal` (array): The input audio signal, normalized to avoid clipping.
- `Fs` (scalar): The sampling frequency of the audio signal (in Hz).
- `depth` (scalar): The maximum delay depth in seconds, controlling the extent of modulation.
- `rate` (scalar): The low frequency oscillator (LFO) rate in Hz, determining the speed of modulation.
- `wetDryMix` (scalar): The mix ratio between the processed (wet) and original (dry) signal. Valid range: $[0, 1]$.

3.3 Output

The function returns the following output:

- `chorusSignal` (array): The audio signal with the applied chorus effect.

3.4 Algorithm Description

The chorus effect is implemented as follows:

1. The input signal is normalized to prevent clipping.
2. The maximum delay depth is converted into samples.
3. A sinusoidal low-frequency oscillator (LFO) is generated to modulate the delay time.
4. A delay buffer is created to store the delayed samples.
5. For each sample in the input signal:
   - The current delay time is determined by the LFO's output.
   - The delayed sample is retrieved from the buffer if sufficient samples exist; otherwise, it is set to zero.
   - The wet and dry signals are mixed based on the `wetDryMix` parameter.
6. The final signal is normalized to avoid clipping.

3.5 Mathematical Representation

The modulated delay time is defined as:

$$\text{currentDelay}(n) = \frac{1 + \sin(2\pi n/\text{lfoSamples})}{2} \cdot \text{depthSamples}.$$

The final signal is computed as:

$$\text{chorusSignal}(n) = \text{wetDryMix} \cdot \text{delayedSample}(n) + (1 - \text{wetDryMix}) \cdot \text{inputSignal}(n).$$

4 Echo Effect Implementation

4.1 Function Overview

The `customDelay` function implements an echo effect for audio signals. Echo is achieved by adding delayed repetitions of the original signal, attenuated by a decay factor. This function allows customization of the delay time, decay intensity, and wet/dry mix ratio.

```
function echoSignal = customDelay(inputSignal, Fs, delayTime, decayFactor, wetDryMix)
```

4.2 Input Parameters

The function accepts the following input parameters:

- `inputSignal` (array): The input audio signal, normalized to avoid clipping.
- `Fs` (scalar): The sampling frequency of the audio signal (in Hz).
- `delayTime` (scalar): The delay time in seconds between echoes.
- `decayFactor` (scalar): The attenuation factor for each successive echo. Valid range: $[0, 1]$.
- `wetDryMix` (scalar): The mix ratio between the processed (wet) and original (dry) signal. Valid range: $[0, 1]$.

4.3 Output

The function returns the following output:

- `echoSignal` (array): The audio signal with the applied echo effect.

4.4 Algorithm Description

The echo effect is implemented as follows:

1. The input signal is normalized to prevent clipping.
2. The delay is calculated in samples based on the specified `delayTime` and sampling frequency (`Fs`).
3. The output buffer is initialized with a length equal to the input signal length plus the delay.
4. For each sample in the input signal:
    - The current sample is added to the output signal.
    - If the delay has elapsed, an attenuated copy of the delayed sample is added back to the output buffer.
5. A wet/dry mix is applied to combine the echoed signal with the original signal.
6. The final signal is normalized to ensure no clipping occurs.

4.5 Mathematical Representation

The echo signal is computed iteratively as:

$$\text{echoSignal}[n] = \begin{cases} \text{inputSignal}[n], & \text{if } n \leq \text{delaySamples} \\ \text{inputSignal}[n] + \text{decayFactor} \cdot \text{echoSignal}[n - \text{delaySamples}], & \text{if } n > \text{delaySamples}. \end{cases}$$

The final signal is blended as:

$$\text{finalEchoSignal} = \text{wetDryMix} \cdot \text{echoSignal} + (1 - \text{wetDryMix}) \cdot \text{inputSignal}.$$

5 Distortion Effect Implementation

5.1 Function Overview

The `customDistortion` function implements a distortion effect for audio signals. Distortion introduces harmonic overtones by clipping the audio waveform at a specified threshold, which emulates the behavior of overdriven amplifiers or analog circuits.

```
function distortionSignal =
customDistortion(inputSignal, gain, clipLevel, wetDryMix)
```

5.2 Input Parameters

The function accepts the following input parameters:

- `inputSignal` (array): The input audio signal, normalized to avoid clipping.
- `gain` (scalar): The gain applied to the input signal before clipping. Higher values increase distortion intensity.
- `clipLevel` (scalar): The threshold for clipping the signal. Values above this level are clipped.
- `wetDryMix` (scalar): The mix ratio between the processed (wet) and original (dry) signal. Valid range: $[0, 1]$.

5.3 Output

The function returns the following output:

- `distortionSignal` (array): The audio signal with the applied distortion effect.

5.4 Algorithm Description

The distortion effect is implemented as follows:

1. The input signal is normalized to avoid initial clipping.
2. A gain is applied to the signal to amplify it.
3. The signal is clipped to the specified `clipLevel`, ensuring values exceeding this threshold (positively or negatively) are limited.
4. The clipped signal is normalized again to ensure it stays within the valid audio range.
5. A wet/dry mix is applied to blend the distorted signal with the original signal.

5.5 Mathematical Representation

The clipping operation is defined as:

$$\text{distorted}(n) = \begin{cases} \text{clipLevel}, & \text{if gain} \cdot \text{inputSignal}(n) > \text{clipLevel} \\ -\text{clipLevel}, & \text{if gain} \cdot \text{inputSignal}(n) < -\text{clipLevel} \\ \text{gain} \cdot \text{inputSignal}(n), & \text{otherwise.} \end{cases}$$

The final signal is computed as:

$$\text{distortionSignal} = \text{wetDryMix} \cdot \text{distorted} + (1 - \text{wetDryMix}) \cdot \text{inputSignal}.$$

6 Tremolo Effect Implementation

6.1 Function Overview

The `customTremolo` function implements a tremolo audio effect from scratch. Tremolo is a modulation effect that varies the amplitude of the input audio signal using a low-frequency oscillator (LFO). This creates a rhythmic pulsating sound.

```
function tremoloSignal = customTremolo(inputSignal, Fs, rate, depth, wetDryMix)
```

6.2 Input Parameters

The function accepts the following input parameters:

- `inputSignal` (array): The input audio signal, normalized to avoid clipping.
- `Fs` (scalar): The sampling frequency of the audio signal (in Hz).
- `rate` (scalar): The modulation rate of the tremolo effect (in Hz).

- `depth` (scalar): The depth of modulation, determining the intensity of the effect. Valid range: $[0, 1]$.
- `wetDryMix` (scalar): The mix ratio between the processed (wet) and original (dry) signal. Valid range: $[0, 1]$.

6.3 Output

The function returns the following output:

- `tremoloSignal` (array): The audio signal with the applied tremolo effect.

6.4 Algorithm Description

The tremolo effect is implemented as follows:

1. The input signal is normalized to avoid clipping.
2. A time vector `t` is created based on the length of the signal and the sampling frequency.
3. A low-frequency oscillator (LFO) is generated using a sine wave. The LFO modulates the amplitude of the signal based on the specified depth.
4. The input signal is multiplied by the LFO to apply the modulation.
5. A wet/dry mix is applied to blend the modulated signal with the original signal.
6. The output signal is normalized again to ensure no clipping occurs.

6.5 Mathematical Representation

The amplitude modulation is performed using the following equation:

$$\text{modulated}(n) = \text{inputSignal}(n) \cdot [1 - \text{depth} \cdot (0.5 + 0.5 \cdot \sin(2\pi \cdot \text{rate} \cdot t))]$$

where t is the time vector and n is the sample index.

The final signal is computed as:

$$\text{tremoloSignal} = \text{wetDryMix} \cdot \text{modulated} + (1 - \text{wetDryMix}) \cdot \text{inputSignal}$$