A Pitch Shifting Reverse Echo Audio Effect

Jingjie Zhang, Ziheng Chen
Center for Computer Research in Music and Acoustics (CCRMA), Stanford University
660 Lomita Drive, Stanford, CA 94305, USA
[jingjiez|zihengc]@ccrma.stanford.edu

***Abstract*—Reverse echo is a type of audio effect that delays the input audio samples and plays them back in a reverse order. Pitch shifter effect, on the other hand, shifts the pitch of the input audio signal by time-stretching and re-sampling. In this work, the reverse echo effect is combined with the pitch shifter to create harmonic reversed echoes. The algorithm is implemented and tested on iOS platform in real time.**

I. Introduction

DIGITAL audio effects are algorithms that adjust and enhance the output audio signal of a musical instrument. Reverse echo, also known as backwards echo, is an audio effect that produces echoes of the input audio recordings that are played backwards, which is widely used by various modern musicians. Like many other time-based audio effects, reverse echo is based on delay line manipulations, which involves the control of read and write pointers of the delay line, however, the write pointer in the reverse echo effect is handled in a slightly different way.

Real-time pitch shifting is a classic topic in audio signal processing. Different methods like Synchronous Overlap-Add (SOLA), Sinusoidal Modeling, and Phase Vocoder are widely-used. Phase-vocoder-based pitch shifting algorithm utilizes phase insensitivity of human hearing, which decompose each audio block into frequency bands and manipulate their phase to keep audio continuity. It keeps most of the original audio information and performs well for music signals.

In this project, we integrate the phase-vocoder-based pitch shifter algorithm into the reverse echo effect so that the original audio input can be mixed with harmonic reversed echoes and hence, enrich the musicality of the audio output.

The rest of this article is organized as follows: Section II gives a detailed review of the basic delay line operations and two reverse echo algorithms. In Section III, the principles and procedures of the pitch shifter algorithm are illustrated. Section IV demonstrates the system structure and implementation scheme, while the results are presented and discussed in Section V. Finally, Section VI summarizes the project and proposes several future directions.

II. Reverse Echo Algorithm

In this section, the basic concept and operations of digital delay lines are reviewed. On the basis of that, two different reverse echo effect algorithms are derived.

A. Digital Delay Lines and Fixed-Length Delay Systems

A digital delay line is a digital buffer that stores the input data samples for several clock cycles and then pops them to the output. One naive implementation of a N-sample delay system could be a length-M ($M > N$) array with a pointer indicating the read/write position. In each clock cycle, one data sample whose position in the array is referred by the pointer will be sent to the output and then overwrote by the input sample. The pointer will move one step forward in the array at the end of each clock cycle. The length of the data array is set to M so that it is capable of realizing any delay lengths shorter than M. However, when changing the delay length, such a structure will produce discontinuities in the output signal, which is not acceptable in audio effects that are based on delay length manipulations.

To produce a smooth transition between different delay lengths, circular delay lines and interpolation methods are introduced in practical delay line implementations.

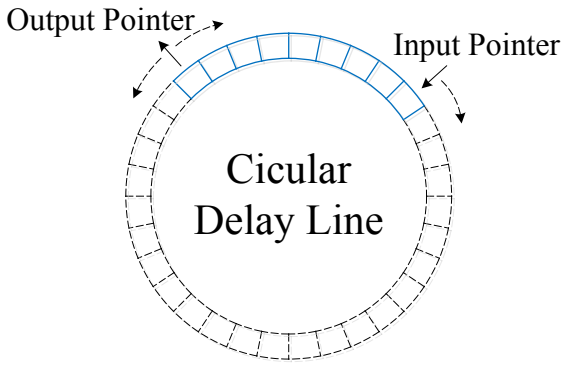

Fig. 1. An example of a circular delay line.

As presented in Fig. 1, a circular delay line, usually very long, has an input pointer that keeps moving one step forward in each clock cycle and an output pointer chasing the input pointer along the buffer. If the distance between the two pointers is fixed, the system becomes a fixed-length delay system, but when the delay length changes, such a structure guarantees that the data samples near the output pointer are all recorded in the same time period and thus will not result in huge discontinuities. In practical audio software, causal smoothing-filters [1] are always applied to the controllers of the parameters, such as delay lengths, to make sure their output curves are continuous. As a result, the movements of the output pointer will always be within adjacent samples in the buffer.

To further achieve the continuous changes in delay length, delay lines must have the capability of dealing with non-integer delay lengths. Hence, in practical delay line implementations, certain interpolation methods are usually applied to the data samples in the buffer to evaluate the data value in-between the sample interval.

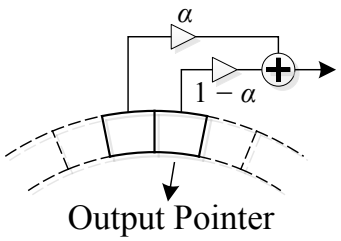

Fig. 2. A linear interpolation filter applied in a delay line.

Among all these interpolation methods, linear interpolation is the most flexible and efficient one. As demonstrated in Fig. 2, to realize a non-integer delay length $N + \alpha$ using linear interpolation, where N is the integer part and α is the fractional part, an interpolation filter is applied to the output sample and its neighbor, which gives an approximation $\hat{x}$ of the output signal x at the non-integer index $n - N - \alpha$:

$$\hat{x}(n-N-\alpha) = (1-\alpha)\cdot x(n-N) + \alpha \cdot x(n-N-1). \quad (1)$$

The relationship in (1) can also be simplified to save one multiplication:

$$\hat{x}(n-N-\alpha) = x(n-N) + \alpha \cdot [x(n-N-1) - x(n-N)]. \quad (2)$$

B. Echo and Reverse Echo Algorithms

As shown in Fig. 3, a simple echo effect can be constructed on the basis of a delay line module. To create decaying echoes of the input signal, the delayed signal is sent back into the delay line after multiplied by a feedback gain that is smaller than 1. The original signal also needs to be mixed with the output signal, otherwise only echoes can be heard.

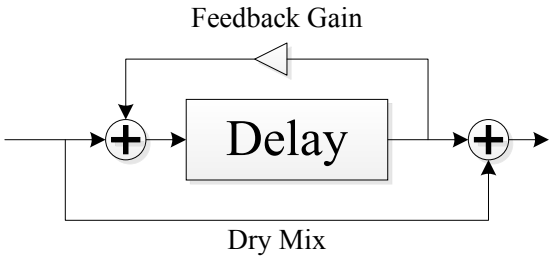

Fig. 3. The signal flow diagram of a simple echo effect.

On the other hand, reverse echo algorithms require a different strategy to operate the delay lines. First of all, to replay the delayed samples backwards, the output position pointer must move in the direction opposite to the input pointer. In addition, since the total length of the circular delay line is finite, the input signal can only be reversed block by block. If the block size is large enough, considerable audio data can be included in each block. This constant block size can also be called the reverse delay length, compared to the "forward" delay length, which in this case keeps changing as the output pointer moves away from the input pointer.

Fig. 4 illustrates the positions of the input and output pointers at some crucial points in the reverse delay process. The input pointer's initial position I_1 corresponds to the first sample in the current block, while the output pointer begins at O_1, which is the last sample in the previous block. As the input pointer keeps moving forward to fill the current block, the output pointer goes backwards to replay the previous audio block in a reverse order. By the end of this stage, the input pointer will stop at I_2, which is the last sample in the current block, and the output pointer will stop at O_2, which corresponds to the first sample of the previous block. In the next clock cycle, the input pointer will move one step forward to I_3 as usual, while the output pointer, instead of moving backwards as expected, will jump to O_3. At this point, the relative positions between input and output pointers is back to the initial case, and the process described above will be repeated. It is not difficult to find that the maximum block size is half the total length of the circular delay line.

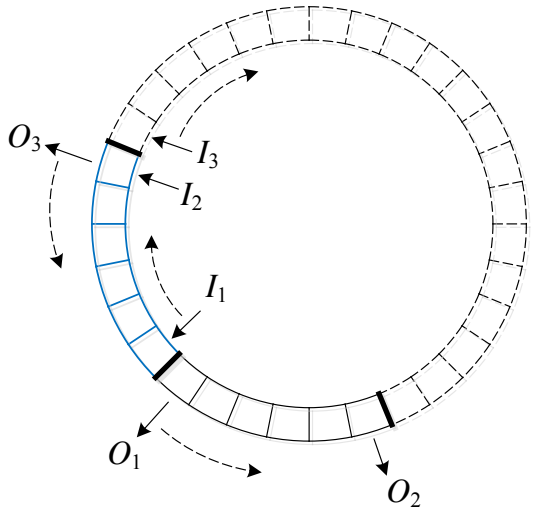

Fig. 4. An example of the input and output pointer positions at different stages in the reverse delay process.

The sudden transition of the output pointer position from O_2 to O_3 will result in clicks in the output audio signal. One way to suppress such clicks is to multiply the output signal with a gain function that decreases to zero as the output pointer moves closer to the boundary. A simple example of such functions can be:

$$G(\frac{Delay}{Delay_{max}}) = 4 \cdot \frac{Delay}{Delay_{max}} \cdot (1 - \frac{Delay}{Delay_{max}}), \quad (3)$$

where $Delay$ is the current delay length defined by the distance between input and output pointers, and $Delay_{max}$ is twice the block size. Therefore, the ratio of $Delay$ to $Delay_{max}$ is always between 0 and 1. Fig. 5 demonstrates the behavior of the gain function $G(x)$ when $0 \leq x \leq 1$, which meets the requirements for the click suppression.

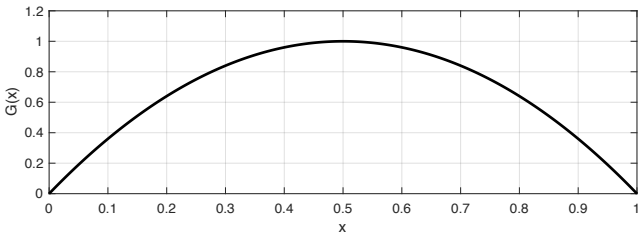

Fig. 5. $G(x) = 4 \cdot x \cdot (1 - x), 0 \leq x \leq 1$.

Replacing the delay module in Fig. 3 with a reverse delay module that carries out the procedures illustrated above will result in a new effect that produces both reverse and forward echoes. As demonstrated in Fig. 6, the input signal is delayed and reversed to create the first echo, however, this reversed echo is sent back into the reverse delay module so that the second echo will be a repeatedly reversed echo and hence, a forward echo. In other words, the output echoes of this system is by turns reversed and forward.

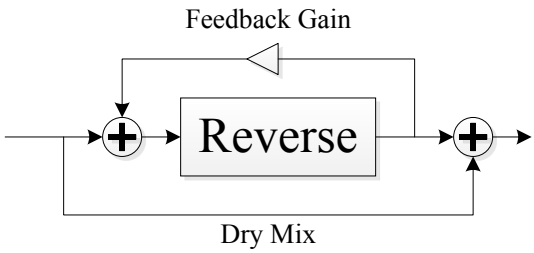

Fig. 6. The signal flow diagram of a reverse and forward echo effect.

To construct a pure reverse echo effect, the reverse delay module is connected with the forward echo system in Fig. 3, so that the first reversed echo will not go back into the reverse delay module again, as shown in Fig. 7. Instead, the first reversed echo is sent into both the output and the forward echo system to produce the following reversed echoes.

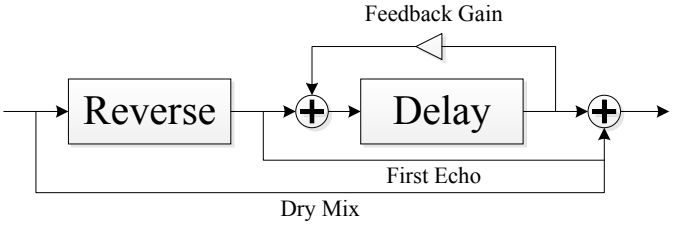

Fig. 7. The signal flow diagram of a pure reverse echo effect.

III. Pitch Shifter Algorithm

Pitch shifter is implemented using real-time phase vocoder algorithm. The input audio signal is first time-stretched and then re-sampled to the original length to generate pitch-shifted output. In this section, we review the basic concept and implementation details of phase vocoder algorithm.

A. Time-Streching

There are mainly 3 stages in using phase vocoder to do time-stretching: analysis, spectral manipulation, and synthesis [2]. Fig. 8 shows the basic workflow of phase vocoder algorithm. Basically, we divide audio signals into segments (with overlap), transform each time-domain segment into frequency domain and modify its phase to keep coherence, and then synthesis these segments with a different step size. The time-stretching ratio is determined by:

$$ratio = h_{syn}/h_{ana}, \tag{4}$$

where h_{syn} is the reconstruction step size in synthesis stage and h_{ana} is the decomposition step size in analysis stage. For example, if h_{syn} is twice as large as h_{ana}, the duration of the audio signal will be doubled.

In the following paragraphs, we will explain in detail two key components used in time-stretching algorithm.

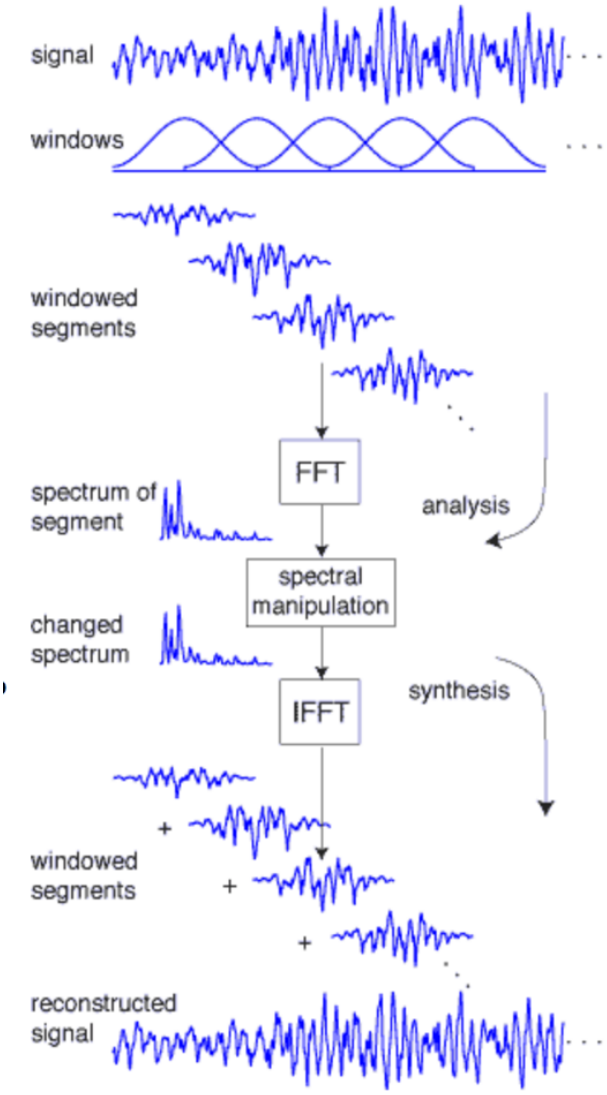

Fig. 8. Phase vocoder workflow [3].

1) Short Time Fourier Transform (STFT) and Constant Overlap-Add Windows: In the analysis stage, we decompose signals into short segments by multiplying a series of overlapped windows. Short Time Fourier Transform is then applied on each windowed segments so that we can manipulate its frequency characteristics at next stage. To perfectly reconstruct the signal at the synthesis stage, the overlapped windows must satisfy the following equation:

$$\sum_{r=-\infty}^{\infty} w(n - rR) = C, \tag{5}$$

where w is the window function, n is the sample number in each window, R is the segment step size, r is the index of each overlapped window, and C is a constant number. The signal can be perfectly reconstructed as long as the overlapped window functions summed up to a constant.

Different window functions may satisfy (5) with different overlapped step size. In our implementation, we choose Hanning Window with 75% overlap-add at synthesis stage, that is, the step size is a quarter length of the window size.

2) Spectral Manipulation: Since the decomposition step size at analysis stage and the reconstruction step size at synthesis stage are different, phase adjustment needs to be applied on each segment before synthesis to keep phase coherence between adjacent segments.

Since different frequency component has different phase shift, we first estimate the frequency components contained in each segment. To estimate frequencies that lies in-between frequency bins, the analysis phase difference in time direction between segments is used. Assume the true value of the frequency we are estimating is $\Omega(rad/s)$, and bin k is the closest bin to Ω, then we have:

$$\phi_k(m+1) - \phi_k(m) + 2\pi L = h_{ana} * \Omega. \tag{6}$$

Note phase values for each segment are calculated by $arctan$ function, where the results are wrapped to the range $(-\pi, \pi]$. In equation (6), $\phi_k(m)$ is the wrapped phase value of bin k in frame number m, $2\pi L$ is an unknown constant representing how much phase is wrapped, and h_{ana} is the decomposition step size (in second) at analysis stage.

The true frequency value Ω can be decomposed to two parts:

$$\Omega = \Omega_k + \Delta_k, \tag{7}$$

where Ω_k is the closest bin frequency to Ω, and Δ_k is the difference between them. Using this property, and assume $\mid h_{ana} * 2\pi\Delta_k < \pi \mid$, we can avoid the influence of the phase-wrapping term $2\pi L$ by modulo 2π to both side of the equation (6). We can get:

$$\Delta_k * 2\pi\Delta_k = (\phi_k(m+1) - \phi_k(m) - h_{ana} * 2\pi\Omega_k)_{mod 2\pi}. \tag{8}$$

We can then calculate Δ_k from equation (8) and get estimated frequency Ω using equation(7)

After estimating each frequency component in a segment, the adjusted phase of this segment is calculated by:

$$\phi'_k(m+1) = \phi'_k(m) + h_{syn} * \Omega, \tag{9}$$

where $\phi'_k(m)$ is the adjusted phase value of bin k in frame number m, h_{syn} is the reconstruction step size in synthesis stage.

B. Re-sampling

After time-stretching, we re-sample the signal to its original length to generate pitch-shifted signal. The re-sample factor is the same as the time-stretching ratio in equation (4). For example, if we time-stretched the signal to twice its original the duration, we then down-sample the time-stretched signal by 2 to restore its original length. The frequency components in the output signal then become doubled, in other words, the pitch is up-shifted by an octave.

IV. Implementation Scheme

The signal flow diagram of the whole system is presented in Fig. 9. The input signal is first pitch shifted and then sent into the reverse echo system to produce harmonic reversed echoes. Both reverse echo algorithms discussed above are embedded into the system so that users are able to select whether the echoes are pure reversed or not.

In the delay line implementation, relative output position is introduced instead of an absolute output position pointer so that the operations on the output pointer in each clock cycle can be avoided. In other words, the output position is actually determined by subtracting the non-integer delay length from the current position of the input pointer in the circular buffer. Such an implementation strategy is perceptually more close to the concept of a delay line, especially with a non-integer delay length, while managing a separated absolute output pointer might cause some mistakes if not carefully handled.

It is also worth mentioning that the quality of the pitch shifting effect based on the phase vocoder strongly depends on the audio buffer size. Only with a buffer size larger than 2048 samples can the system produce a decent pitch shifted output signal. Thus, to reduce the overall latency of the system, while using a small 512-sample audio buffer, an internal buffer with a size of 4096 samples is introduced so that the pitch shifter algorithm is carried out every 8 audio blocks, producing the output signal for the next 8 audio output blocks.

In addition to the audio effects described in the previous sections, a simple Feedback Delay Network (FDN) reverberation effect [4] is applied to the final output of this system to improve the quality of the whole effect. The FDN reverberation effect uses multiply short delay lines to simulate the early reflections of an original sound in an acoustic space and introduces a lossless feedback matrix to realize the reverberation process.

V. Results

The guitar effect system illustrated in the previous sections is implemented using Swift and C++ on the iOS platform. In this section, the output signals of the reverse echo and pitch shifter effects are evaluated separately, and the design and implementation of the graphical user interface is also presented.

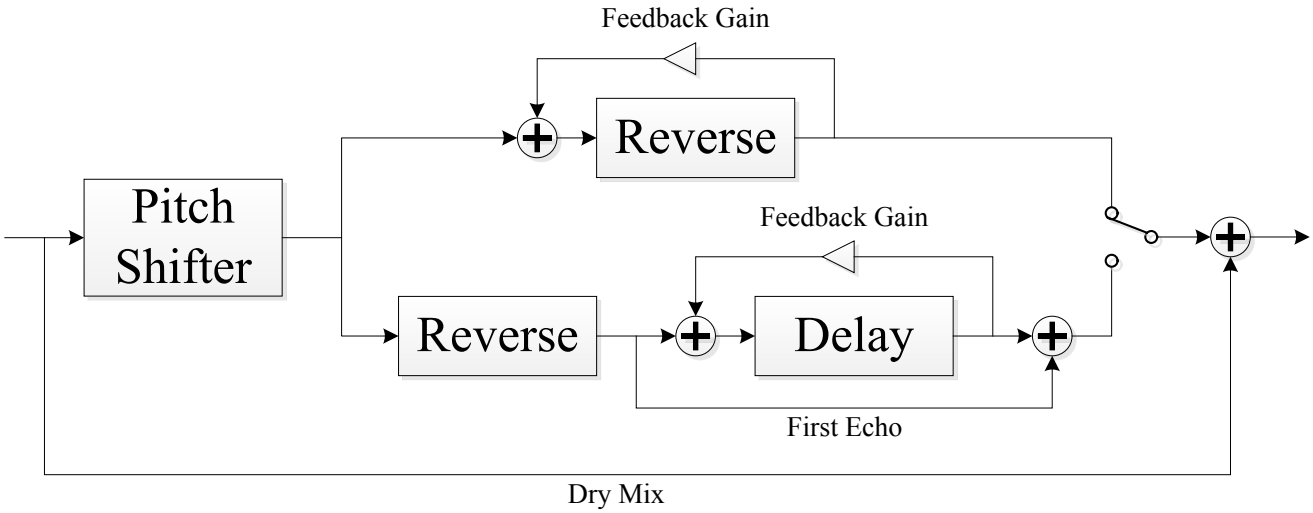

Fig. 9. The system diagram of the whole pitch shifting reverse echo effect.

A. Reverse Echo Effects

Fig. 10 shows the output signal of a reverse and forward echo effect with a feedback gain of 0.8 and a block size of 11025 samples under the sample rate of 44.1 kHz. According to the result, the decaying echoes created by the effect are by turns reversed and forward.

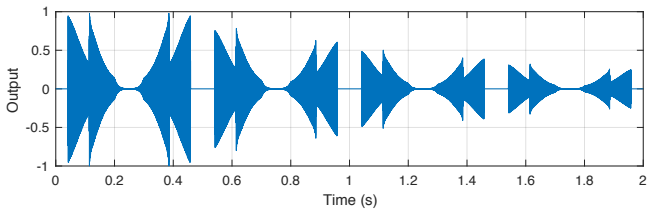

Fig. 10. The output signal of a reverse and forward echo effect.

The output signal of a pure reverse echo effect is also presented in Fig. 11 with similar parameter settings, while in this case, the output echoes are all reversed.

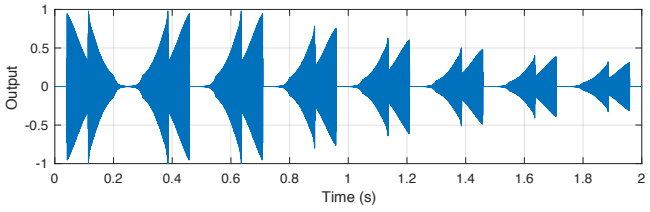

Fig. 11. The output signal of a pure reverse echo effect.

B. Pitch Shifter Effect

Fig 12 shows the result of a pure sinusoid processed by pitch-shifting algorithm. The top graph is the original signal, which is a 4-second long, 5Hz sinusoid sampled at 4KHz. The middle graph illustrates the result of the original signal time-stretched to twice the length. The bottom graph shows the result of re-sampling the time-stretched signal to restore its original length, in which case the frequency becomes twice the original frequency (pitch is scaled up by an octave).

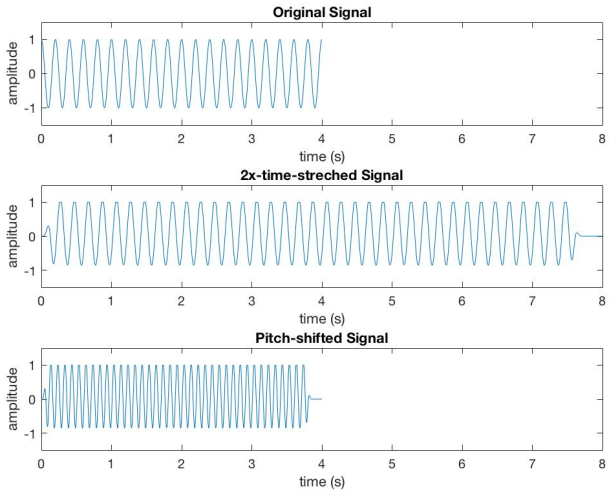

Fig. 12. A pure sinusoid time-stretched and pitch-shifted

C. Graphical User Interface Design

The user interface should be capable of controlling certain parameters such as the delay length of the echoes, the ratio of the wet signal mixed in the total output and the feedback gain. As presented in Fig. 13, three sliders controlling such parameters are included in the GUI implementation on iOS platform. In addition, the users are able to switch between the pure reverse echo effect and the reverse and forward echo effect using the ”Pure Reverse” switch. It is also important that the whole effect can be disabled or enabled by the main switch at the bottom.

VI. Conclusion

In this project, the phase vocoder pitch shifting algorithm is combined with two reverse echo effect to form a pitch

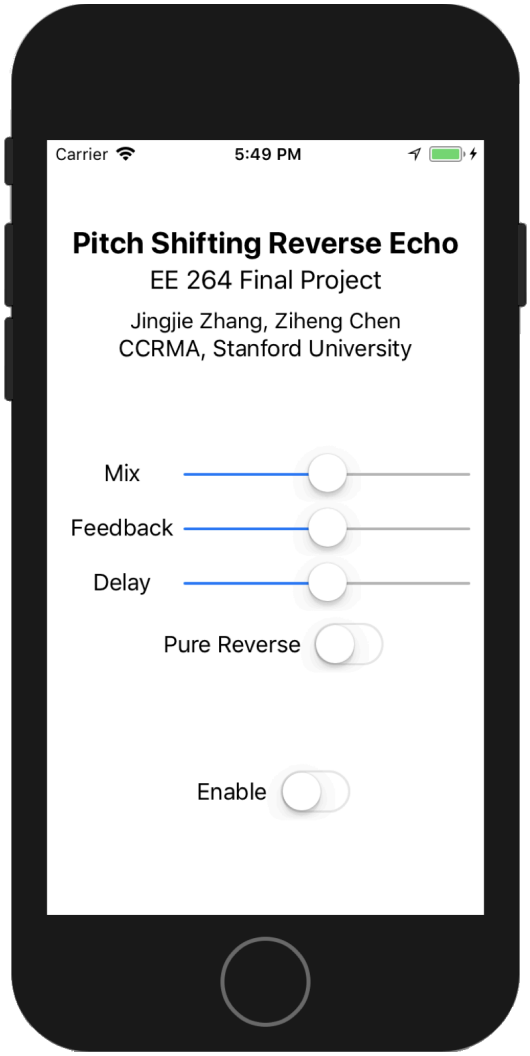

Fig. 13. The GUI design on iOS platform.

shifting reverse echo audio effect system. An internal buffer is introduced to the pitch shifter so that the high quality of the pitch shift can be maintained in a low latency audio system with a small audio buffer size.

Currently, the input audio signal is shifted up by one octave and sent into the reverse echo system. In the future, another signal path can be added to the system where the input signal is shifted up by two octave and sent into another reverse echo system with a different delay length to add more harmonics to the total output signal.

In addition, the phase-vocoder-based pitch shifter module implemented in this project still needs a relatively large buffer to maintain the quality of the pitch shifting, while it is possible to use more advanced algorithms that can work well with a buffer size of 512 samples or less. With a more efficient pitch shifter module, it is also interesting to experiment with the location of this module in the echo effect system and see its impact to the audio effect.

VII. Acknowledgements

Thanks to Professor Fernando Mujica for his supportive guidance throughout this project.

References

[1] J. O. Smith. Convolution Example 2: ADSR Envelope. *Mathematics of the Discrete Fourier Transform (DFT) with Audio Applications*. online book, 2007 edition, https://ccrma.stanford.edu/~jos/mdft/Convolution_Example_2_ADSR.html, accessed Mar. 2018.

[2] J. L. Flanagan and R. M. Golden. Phase vocoder. in *The Bell System Technical Journal*. vol. 45, no. 9, pp. 1493-1509, Nov. 1966.

[3] William A. Sethares. A Phase Vocoder in Matlab. http://sethares.engr.wisc.edu/vocoders/phasevocoder.html, accessed Mar. 2018.

[4] J. O. Smith. FDN Reverberation., *Physical Audio Signal Processing*. online book, 2010 edition, https://ccrma.stanford.edu/~jos/pasp/FDN_Reverberation.html, accessed Mar. 2018.