EXTENDED PLUCKED STRING MODEL FOR ELECTRIC GUITAR SOUND SYNTHESIS

Junhao Wang
McGill University
MUMT 618 Final Project

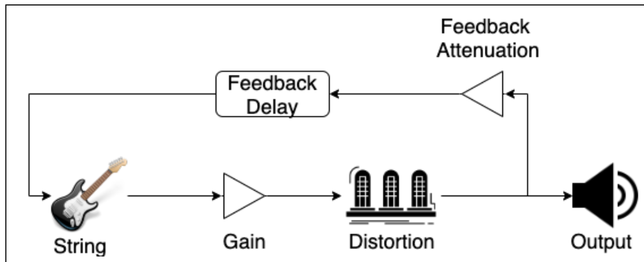

Figure 1. Diagram of the implemented system

1. INTRODUCTION

Karplus-Strong algorithm, initially proposed in [3], is a simple and efficient algorithm for synthesizing real-sounding plucked string sound. The basic design of this algorithm makes use of a delay line and a low-pass filter. The delay line length controls the pitch and the low-pass filter ensures that higher frequencies decay faster than lower ones. Based on this simple design, [2] and [4] proposed various extensions and improvements to the original Karplus-Strong algorithm, which improved its usability and the quality of generated sound.

The main objective of this project is to explore and implement the extensions proposed by [4], which extends Karplus-Strong algorithm to simulate specifically the plucked string sound of an electric guitar. Beyond the basic string model, distortion and feedback effects are also modeled and incorporated into the system, which helps making a more convincing and interesting electric guitar sound. The model is prototyped in MATLAB and an interactive demo is written in Python [1] .

2. OVERVIEW

An overview of the system implemented in this project is shown in Figure 1. The design is similar to the actual set-up that a guitar player may use in real life. The string models simulate the guitar itself. The distortion unit processes and introduces non-linearity to the signal generated by the string models, just like a real amplifier or distortion pedal would do in a signal chain. After distortion, the signal goes through a feedback unit which simulates the high-pitch growing oscillation that happens when the electric guitar is placed very close to the speaker. Each component will be discussed in more detail in the next section.

[1] Source code available at https://github.com/jwang44/KS-extended

3. PROJECT DESIGN

Most part of this project is designed according to [4] and implemented in MATLAB. An interactive demo is written in Python with a simple graphical user interface.

3.1 String model

3.1.1 Loop Filter

The basic design of a Karplus-Strong string model consists of a delay line and a low-pass filter. The delay line length N is set to obtain the desired fundamental frequency, as given by the equation

$$N = f_s/f_0 \tag{1}$$

where f_s denotes the sampling rate and f_0 denotes the fundamental frequency. For the low-pass filter, a 2-point averaging filter is used in the original design. Although this filter has linear phase and monotonically decreasing amplitude response, it does not allow adjustment to decay rates for different frequency regions. To address this issue, a 3-point averaging filter is used instead.

$$y_n = a_0 x_n + a_1 x_{n-1} + a_0 x_{n-2} \tag{2}$$

It has an amplitude response of

$$|H_1(w)| = |2a_0 \cos w + a_1| \tag{3}$$

If we set $a_1 \geqslant 2a_0 \geqslant 0$, this response is monotonically decreasing. Linear phase is guaranteed by the symmetric nature of its transfer function. Moreover, by adjusting a_0 and a_1, different magnitude response can be obtained, as shown in Figure 2, which enables an extra measure of timbre control.

One problem with this design is that with some choices of coefficients, the dc component may ends up with a gain equal to or higher than 1. In this case, the dc component may not decay as desired and may even grow stronger. This would cause a click at the end of the note. To remove the undesired dc component, two methods are used.

3.1.2 Dc-blocking

Two approaches are taken to eliminate the potential dc component. Considering the source of dc component, the first approach is to remove the dc component from the initial random noise used to excite the string. This can be easily done by subtracting the mean value of the noise vector before putting it into the delay line. During experiment,

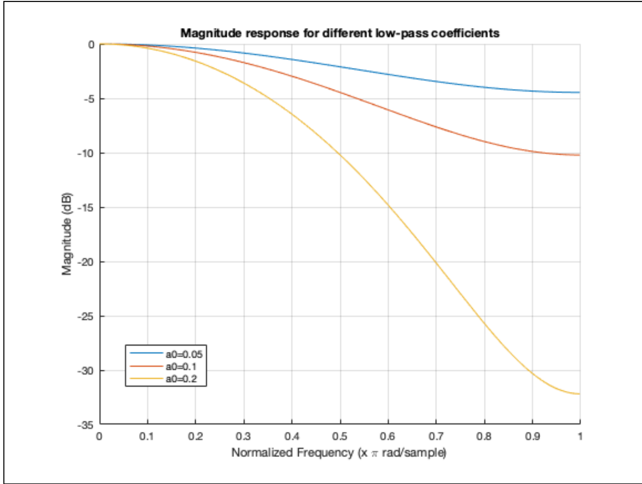

Figure 2. Magnitude response of the loop filter with different coefficients

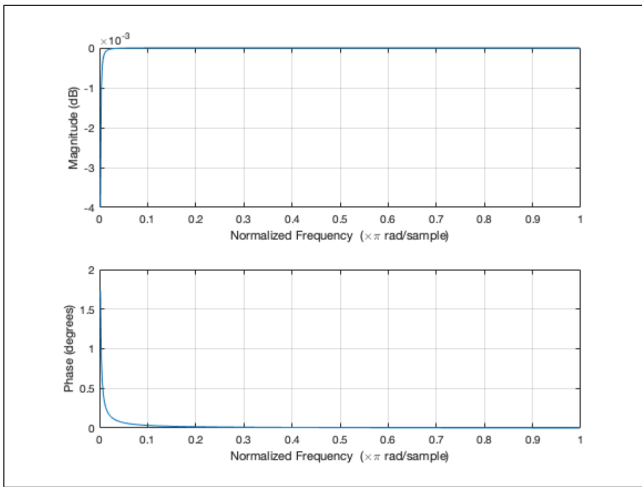

Figure 3. Frequency response of the dc-blocking filter

this was proven not sufficient. Therefore a dc-blocking filter is incorporated into the loop. In this case, a first-order high-pass filter is used.

$$y_n = \hat{a}_0 x_n + \hat{a}_1 x_{n-1} + b_1 y_{n-1} \tag{4}$$

This dc-blocking filter has zero response at dc and almost no effect on frequencies at or higher than the fundamental, if the coefficients are chosen properly as given in [4]

$$\begin{aligned} \hat{a}_0 &= \tfrac{1}{1+\omega_{co}/2} \\ \hat{a}_1 &= -\hat{a}_0 \\ b_1 &= \hat{a}_0 \left(1 - \omega_{co}/2\right) \end{aligned} \tag{5}$$

The cut-off frequency is denoted by ω_{co}, which should be set significantly lower than the fundamental frequency. In this implementation, $\omega_{co} = f_0/10$ is used. Its frequency response is shown in Figure 3.

3.1.3 Exact Tuning

One major drawback of the basic Karplus-Strong algorithm is that it does not allow a continuous range of frequency. As delay line length can only be an integer, it can

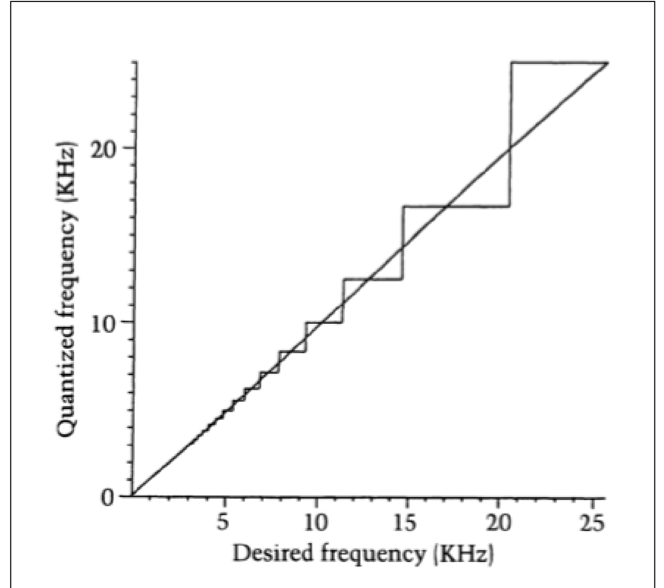

Figure 4. Desired pitch versus resulting pitch for a 50KHz sampling rate [2]

only generate notes with fundamental frequencies whose period is an integer multiple of the sampling period. As demonstrated in Figure 4 from [2], although we desire a continuous range of frequencies, the resulting frequencies are quantized to the nearest level. The difference between two adjacent levels gets larger as desired frequency gets higher and delay line gets shorter. For low pitches, this does not cause noticeable inaccuracy in tuning, but for higher pitches, the difference becomes innegligible.

To address this issue, [2] and [4] both proposed the idea of using interpolating delay line, which allows a fractional delay line length. [2] introduced a first-order all-pass filter,

$$y_n = C x_n + x_{n-1} - C y_{n-1} \tag{6}$$

where C is the all-pass coefficient that can be computed with the fractional part of the delay denoted by Δ.

$$C = \frac{1-\Delta}{1+\Delta} \tag{7}$$

This filter introduces a small delay while keeping the loop gain unchanged. But as suggested in [4], this all-pass filter does not satisfy linear phase. As shown in Figure 5, the phase delay of the all-pass filter is frequency dependent. Δ is best kept within the range $0.3 \leqslant \Delta \leqslant 1.3$ to achieve an acceptable close-to-linear phase response. If Δ goes outside this range, the non-linearity in phase response can lead to slightly different loop delays for the fundamental and its harmonics. Consequently, the harmonics are perceived slightly out-of-tune with the fundamental. In this project, linear interpolation is used instead of all-pass interpolation. Linear interpolation is implemented as a two-point averaging filter which is similar to the ones used in both the basic design [3] and extended Karplus-Strong algorithm [2]. In Jaffe and Smith's extensions, it's used for achieving desired decay rate, while in this implementation it's used only for addressing fractional delay length. This

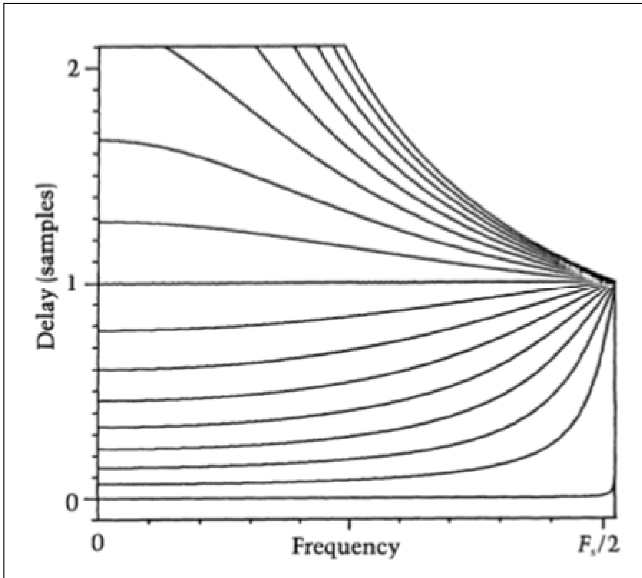

Figure 5. Phase delay for the fine-tuning all-pass filter [2]

2-point averaging filter is not always linear-phase, but as suggested in [4], it's generally closer to linear phase than the all-pass filter. Using linear interpolation instead of all-pass interpolation also benefits glissandi.

3.1.4 Glissandi

To perform a glissando, the fundamental frequency needs to be adjusted continuously. In an electric guitar context, the idea is used in playing techniques such as bend, slide, vibrato, and whammy-bar dive, which are very popular playing techniques that add expression and emotion to the music. These techniques all cause a gradual change in pitch. As described in the last section, the interpolating delay line allows a continuous range of frequencies, which is key to a smooth glissando. The linear interpolation filter can be represented by

$$y_n = c_0 x_n + c_1 x_{n-1} \tag{8}$$

To perform a glissando, coefficients c_0 and c_1 are gradually adjusted. For instance, when a string bend is simulated, the pitch should gradually go up. The delay length should decrease. In this case, c_0 is increased until it hits the upper boundary, where $c_0 = 1$ and $c_1 = 0$. Then delay line length is decreased by one sample and c_0 is set back to 0. As shortening the delay line by one sample is equivalent to having $c_1 = 1$ in the filter, there is no glitch in this process.

In using all-pass filter, the delay length can be adjusted in a similar way. However, due to the non-linear nature of its phase response, the all-pass filter works well only within a certain delay range, i.e. $0.3 \leqslant \Delta \leqslant 1.3$, where Δ denotes the fractional part of the delay. To make sure that its coefficients always stay in this range, as soon as Δ goes below 0.3, one sample should be subtracted from delay line length and added to Δ, this sudden change in coefficients would produce a glitch and thus affect the smoothness of glissandi.

While linear interpolation does work as advertised in [4], its low-pass nature should not be neglected. Choosing between all-pass and linear interpolation is a trade-off between linear phase and constant loop gain. By incorporating linear interpolation into the loop, the signal is put through an extra low-pass filter, which attenuates the high frequency harmonics. It is mentioned by [4] that when using linear interpolation, its magnitude response should be taken into account, and the coefficients in the loop filter should be adjusted accordingly. However, there is no simple way to implement this other than tuning the coefficients for each fundamental frequency. In this project, the linear interpolation is simply cascaded with the loop filter. This leads to extra attenuation for high frequencies, which is one of the limitations of this model.

3.1.5 Tone Control

Plucked string sound synthesized by Karplus-Strong algorithm is considered very realistic. But when compared with a real guitar string, the synthesized sound is often much brighter than the real guitar sound. To further manipulate the timbre of synthesized sound, another extension is added to control the initial harmonic contents. The initial random values are passed through another 3-point averaging filter before being put into the delay line. The filter can be represented as

$$y_n = a(x_n + x_{n-1} + x_{n-2}) \tag{9}$$

The coefficient a is set to $|2\cos\omega_0 + 1|^{-1}$ so that this filter only reduces the amplitudes of harmonics and not the fundamental. Harmonic contents are reduced each time the initial random values are passed through this filter. More passes result in a warmer string sound. Adjusting this achieves an effect somehow similar to adjusting the tone control on an actual electric guitar.

3.2 Distortion

Distortion is originated when the guitar amplifier is turned up so much that circuit limitation is reached and clipping or other non-linear behavior starts to occur. This non-linearity produces a unique sound that is rich in harmonics. For a more vivid and interesting guitar sound, a distortion unit is incorporated into the loop in this implementation. For simplicity, distortion is implemented as a simple non-linear function applied to samples output by the string model. In real amplifiers or distortion pedals, the non-linear characteristics are much more complicated than that.

The output signal of the string model is multiplied by a gain parameter before being sent into the non-linear distortion unit. Soft-clipping, which is somehow similar to the behavior of tube amplifiers, is used in this project. As opposed to hard-clipping, soft-clipping produces a relatively wide range of distortion, as we have more control over intermediate sound levels between 0 and the threshold. Soft-clipping function is represented as follows

$$f(x) = \begin{cases} 2/3; & x \geqslant 1 \\ x - x^3/3; & -1 < x < 1 \\ -2/3; & x \leqslant -1 \end{cases} \tag{10}$$

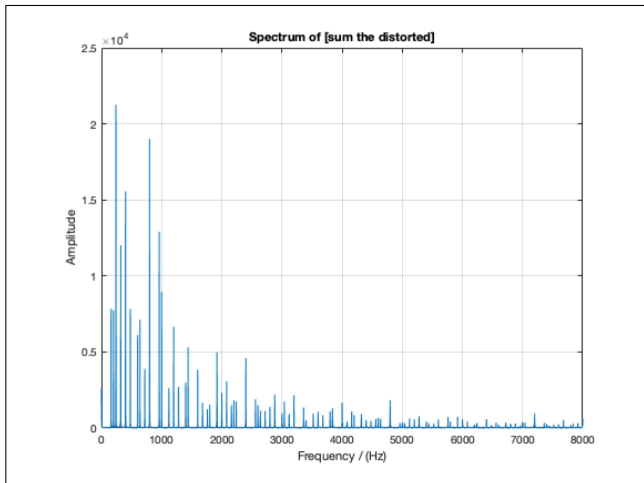

Figure 6. Spectrum resulted from summing the individually distorted notes

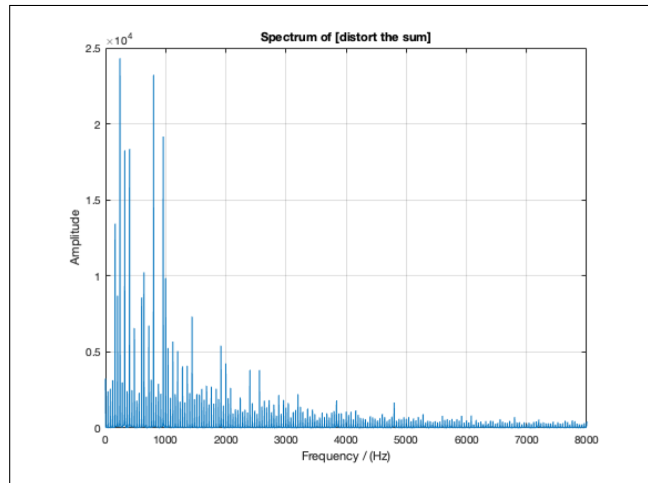

Figure 7. Spectrum resulted from distorting the sum of individual notes

In this project, the output is taken only from the distortion unit, instead of a combination of clean and distorted signal as used in [4]. As we maintain $2a_0 + a_1 = 1$ in the loop filter, the signal multiplied by a gain of 1 does not exceed the non-linear threshold, so it does not get clipped. Passing through the intermediate non-linear region does not make significant difference to the sound. It is thus considered the clean signal. To get a distorted signal, gain parameter is set higher than 1. This simplification reduces the number of parameters and makes it easier to adjust the timbre. The thresholds in the non-linear function also help limit the amplitude of signal when feedback effect is used.

In implementing the distortion, plucking a single note is relatively easy to implement. When synthesizing chords, two approaches are explored. Intuitively, a chord played with distortion effect can be simulated as a combination of several distorted notes. However, this is not how a real electric guitar works. In a real guitar, simultaneous notes are summed together before distortion is added. If we consider the distortion effect as a transfer function, the output should have the same period as input. When we input two notes with fundamental frequencies f_1 and f_2 simultaneously, the output will contain components at the greatest common divisor of f_1 and f_2, and its harmonics, which do not exist in the input signal. These new components play an important part in what we hear in a real distortion. This is different from summing individually distorted notes, where the result only contains the fundamentals and harmonics of each note added by distortion. Figure 6 and Figure 7 shows the difference in spectra of the same chord synthesized using these two approaches. It can be seen that distorting the sum produces a more harmonically-rich sound.

3.3 Feedback

Feedback is originated when an electric guitar is placed very close to the amplifier. Energy from the amplifier (or the speaker cabinet, to be precise) hits the guitar body, neck, and strings. Motion of the strings is picked up by the pick-ups and then amplified by the amplifier. They form a positive feedback loop that produces the feedback effect. For feedback to occur, there must be enough gain in the loop, that is, the player must play loud enough and the guitar must be placed close enough to the speaker cabinet.

In the implementation for this project, a separate delay line is used to simulate feedback delay. The length of feedback delay controls the frequency of the feedback sound. Output samples are attenuated and then written into the feedback delay line and eventually added back to the delay line in the string model. For the feedback effect to work properly, there must be sufficient gain in the loop so that the feedback sound can occur and grow. At the same time, the output signal must be significantly attenuated before adding back to input, otherwise the feedback sound would grow too fast.

4. IMPLEMENTATION DETAILS

The project is first implemented in MATLAB and then migrated to Python for building the interactive demo. The main algorithm, including the string model, *dc*-blocking, tone control, distortion, and feedback, is written as a function in both languages.

4.1 MATLAB Implementation

The main algorithm is written in *pluck.m*, and functionized in *pluck_func.m*. There is a separate MATLAB script *load_midi.m* for synthesizing audio from MIDI files. That script makes use of miditoolbox [5], which is a dedicated library of MATLAB functions for analysing MIDI files. The *pluck_func* function is called in *pluck_chord.m* and *load_midi.m* for synthesizing notes. The distortion unit is written as a separate function named *non-linear.m*.

4.2 Python Implementation

A more complete implementation is done in the interactive demo. The whole algorithm is implemented in *pluck.py*, which includes 2 functions: *pluck* and *pluck_chord*. These

two functions are called in *interface.py*, which generates the graphical interface. Pitch bend is only implemented in Python and can be controlled using the graphical interface. The implementation is otherwise the same as the MATLAB implementation, except the value range of the tone parameter.

5. LIMITATIONS AND FUTURE WORK

The model implemented in this project produces a relatively realistic electric guitar sound. The extensions on karplus-strong algorithm enables more control over the synthesizing process and improves the output quality. The distortion and feedback effect add some more interesting characteristics to the sound. However, this project has some limitations. First, the glissando is implemented in this project, but when synthesizing audio with MIDI files, the pitch bend information in MIDI is ignored. If we can detect and perform pitch bends as given in midi files, the result would be more expressive and useful. Second, for exact tuning, the linear interpolation is simply cascaded with the loop filter. This leads to extra attenuation for high frequencies. Ideally, the attenuation caused by linear interpolation should be compensated by changing the frequency response of the loop filter. Third, the dc-blocking filter used in the string model does not satisfy linear phase and can potentially cause tuning problem for harmonics.

Other than solving the above issues, future development can also focus on applying more complicated design for the distortion unit. Related work [1] [6] [7]on modeling real distortion pedals using virtual analog approach might also work well on synthesized guitar sound.

6. REFERENCES

[1] Eero-Pekka Damskägg, Lauri Juvela, Vesa Välimäki, et al. Real-time modeling of audio distortion circuits with deep learning. In *Proc. Int. Sound and Music Computing Conf.(SMC-19), Malaga, Spain*, pages 332–339, 2019.

[2] David A Jaffe and Julius O Smith. Extensions of the karplus-strong plucked-string algorithm. *Computer Music Journal*, 7(2):56–69, 1983.

[3] Kevin Karplus and Alex Strong. Digital synthesis of plucked-string and drum timbres. *Computer Music Journal*, 7(2):43–55, 1983.

[4] Charles R Sullivan. Extending the karplus-strong algorithm to synthesize electric guitar timbres with distortion and feedback. *Computer Music Journal*, 14(3):26–37, 1990.

[5] Petri Toiviainen and Tuomas Eerola. Midi toolbox 1.1. *URL: https://github. com/miditoolbox/1.1*, 2016.

[6] David T Yeh, Jonathan S Abel, and Julius O Smith. Simplified, physically-informed models of distortion and overdrive guitar effects pedals. In *Proc. of the Int. Conf. on Digital Audio Effects (DAFx-07)*, pages 10–14. Citeseer, 2007.

[7] David Te-Mao Yeh. *Digital implementation of musical distortion circuits by analysis and simulation*. Stanford University, 2009.