MUMT-618 Final Project Report

Virtual Slide Guitar

David G. Smith

12/05/2022

Table of Contents

Introduction

For my final project, I attempted to implement the slide guitar synthesis algorithm as described in the CMJ article by Jyri Pakarinen, Taipo Puputti, and Vesa Välimäki [1]. The development platform was chosen to be Matlab as this would also be an exploration of alternative design implementations to understand the sonic impacts of each component. The output is produced on a sample-by-sample basis to have the highest degree of control possible as well as facilitate isolating issues with a higher degree of precision and accuracy.

In general, the recreating the model was not as straight-forward as the paper would implies Development and verification of the constituent components (i.e. waveguides, pulse generators, etc.) of the synthesis algorithm itself was rather simple. The difficulty came in testing the entire model as a whole, where all the parts need to be synchronized together. Understanding the control signals and properly generating them in a manner which provides musical sounding examples took more effort than initially anticipated. Given that the original paper was focused on a gestural control system, the control signals were defined by the limitations of the motion capture system [1]. Accordingly, the more precise characteristics of the relative string length signal wasn't emphasized. A lot of nuance in the control signals was also obfuscated by over simplification/misleading notation on graphical representations of the synthesis model. Overcoming these issues provided unanticipated obstacles and prevented exploring the unwound string modifications.

Synthesis Model

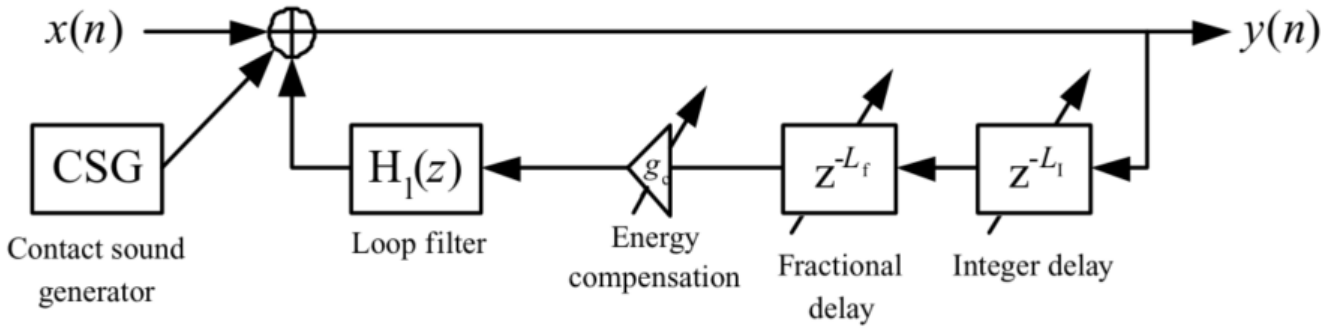

Figure 1: Synthesis Model [1]

The synthesis model consists of two components:

- A single delay line digital waveguide which simulates the string's transverse vibrations
- A contact sound generator model to emulate the string/slide interactions

Each of these has been optimized to help recreate the time-varying aspects associated with slide guitar playing.

String Digital Waveguide

The main modification to the standard approach to a string model is the addition of the energy compensation unit after the fractional delay and before the loop filter. This is mean to help maintain a more perceptually accurate model as when the pitch is varied the total length of the digital waveguide changes. If samples are lost (i.e. the string is shortened) without compensation, then the pitched sound would have an audible volume drop not present in real-world slide guitar playing. Similarly, if the string model is lengthened, samples would be added potentially causing an unnatural boost in volume.

A second change is how the loop-filter has been modified to take into account the relative string length. The full procedure is described in paper from 1998 by one of the authors Vesa Välimäki [2]. To summarize, they recorded a professional guitar player plucking individual notes on each string at all the different fret positions. From these recordings, loop-filter parameters were extracted which closely mimicked the physical losses the different tones experienced. A first-order polynomial fit was then created for the loop-filter's gain and cut-off frequencies which takes the fret number as an argument for interpolation.

Contact Sound Generator

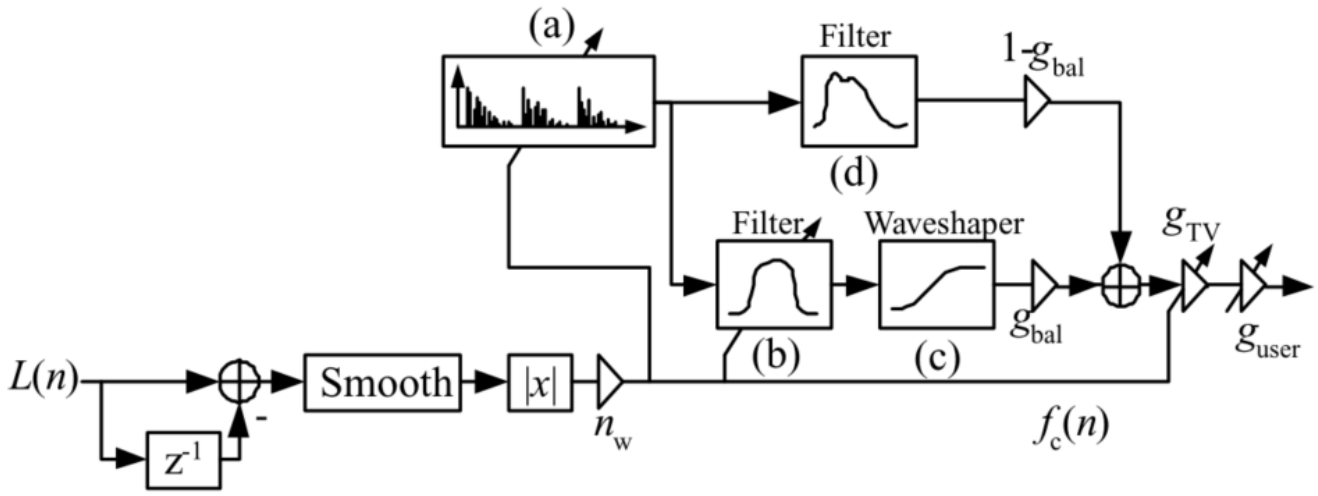

Figure 2: Contact Sound Generator model [1]

The Contact Sound Generator (CSG), in figure 2 above, is a non-standard feature developed for this synthesis model. The lower left branch of the model takes in the relative string length, L(n), performs a first order difference to compute the slide velocity, up-samples this from the control-rate to audio-rate if need be, performs an absolute value as the slide's travel direction doesn't matter, and then scales this by a parameter corresponding to the number of string windings for a particular string. These modifications produce as signal $f_c(n)$ as pictured above, which each of the different adjustable synthesis blocks consumes. In actual practice, the individual blocks consume slightly different data which is derived from the control signal and not $f_c(n)$, a fact which their provided graphic substantially obfuscates.

Ultimately the goal of the CSG module is to recreate a spectrum similar to what is shown in figure 3 below. This spectrogram was generated by damping the transverse vibrations of a string and sliding a

slide across a fixed distance. The slide starts at rest and ends at rest and reaches its highest velocity at the middle point.

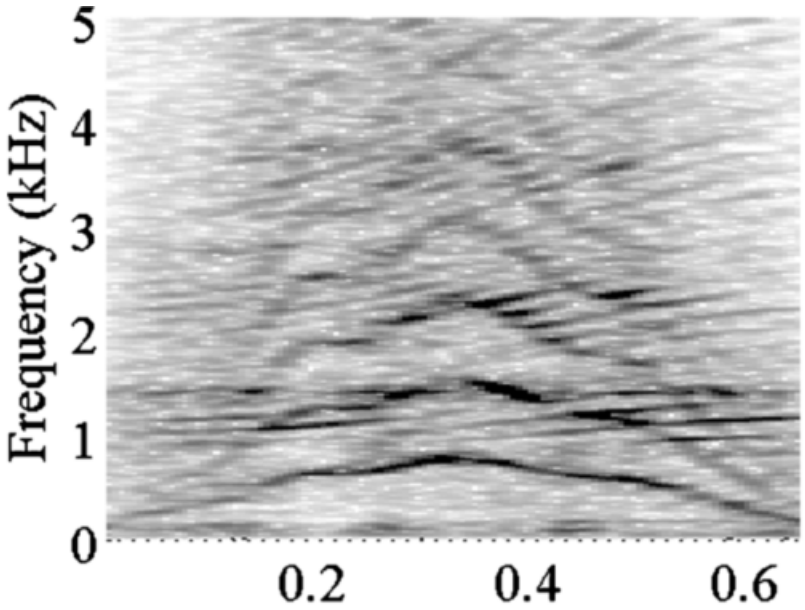

Figure 3: Slide Noise Spectrogram [1]

As is clearly shown, there is a time varying harmonic component whose fundamental frequency is related to the slide velocity. The CSG achieves this via the blocks labeled (b) and (c) where (b) is a 2nd-order resonator who's center frequency is controlled by the slide velocity to produce the fundamental. Block (c) is a hyperbolic tangent function used to generate the harmonics. Block (d) is a static 4^{th} order IIR filter which is used to generate the longitudinal modes which appear as the static frequency content while the slide moves. The source signal, block (a), is an exponentially decaying white noise pulse whose pulse rate is also controlled by the slide velocity.

Implementation

As mentioned before, an attempt to implement the algorithm which allows for calculations on a sample-by-sample basis was made. The fundamental constituent blocks were easy to develop and verify on their own, however the true measure of correct operation is how they operate in a synchronized fashion in relation to the control signal to be able to generate a musically usable sound. This part was not covered in detail in CMJ paper and resulted in quite a few unforeseen roadblocks occurring.

Order of Component Development

The first component I chose to implement was the CSG, due to its complexity as compared to the string model. In hindsight, this was somewhat of a mistake as it was hard to verify the CSG once it was complete beyond a purely functional aspect. For instance, it was quite easy to generate an L[n] signal which caused it to fire pulses at a particular rate and set the resonator's center-frequency to a particular value. However, the realism of this synthesized L[n] was extremely questionable. Ultimately, the CSG

would be operating in conjunction with the string model which is also controlled by L[n]. How the string model responds to different changes in L[n] arguably has a greater impact on the musicality of the resulting sound as there is where the pitch is generated. Correspondingly, the limitations/constraints it imposes on the shape or characteristics of L[n] are of greater importance and the contact sound generator should adapt to those parameters as opposed to the other way around. The overall synthesis context has a large impact in how the components work together to create a musical sound and is just as important as the fundamental synthesis components themselves. Given the string model defines this more so, it would have been better to start by developing that and learning how to control it in order to have the development process be more efficient.

Making L[n] Musical

The goal of any sort of musical instrument is to make a sound which is musical and a synthesizer should be no-exception. However, the ideas which computers operate on and correspondingly how synthesis models are parameterized/controlled do not always align with traditional musical thought. The control signal L[n] here is one such example. It represents the relative string length of the plucked string which is being controlled by the slide. However, guitarists themselves rarely think in terms of relative string lengths. Fret numbers and patterns on the fret board dominate their thinking as these map directly to music theory ideas.

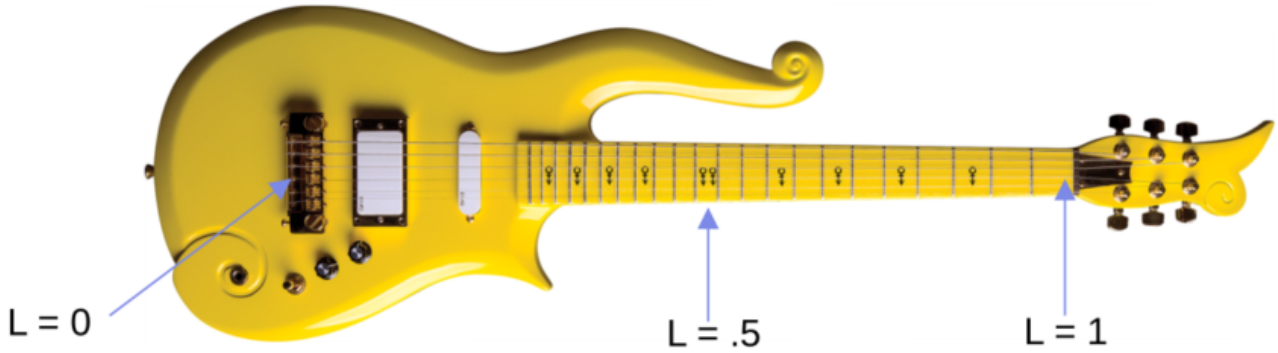

Figure 4: L[n] maps to the interval 0-1. .5 represents an octave.

As shown in figure 4 above, the L[n] signal represents the full-string length, or open string, at value 1. 0 maps to the other extreme of a string of no-length, which technically isn't usable. Arguably the shortest string length possible depends on the limitations of the digital waveguide as well as sampling rate at which the system is run. In order to think more like a guitarist, these values of 0-1 need to be mapped to the fret numbers. This can be achieved by noting that the guitar's frets are spaced according to the 12-TET system as well as examining how L[n] parametrizes the digital waveguide's sounds.

The following relationship expresses how L[n] affects the pitch of the sound synthesized by the digital waveguide:

$$F_{sel} = \frac{F_{open}}{L[n]}$$

F_{sel} is the fundamental of the selected pitch and F_{open} is the fundamental of the open string. Given that 1-fret corresponds to a half-step ratio we can also generate the following equation where $2^{1/12}$ comes from the 12-TET definition of a half-step:

$$F_{sel} = F_{open} * 2^{Fret\ number/12}$$

From these two equations, we can derive expressions to determine L from the fret number and vice versa:

$$L = 2^{-Fret\ number/12} \quad \text{and} \quad Fret\ Number = -12 * \log_2(L)$$

This allows a programmer to much more easily develop L[n] signals in a more guitarisitic thought process. It is much easier to now specify a slide from the 3rd fret to the 5th fret or other similar ideas. Given that the original paper developed this synthesis model to be used with a gestural control system, it would make sense that the original authors didn't need to specify the control signal in this manner as this information was already encoded in the motions they used to control the synth.

Controlling 2nd-Order Resonator

Another issue regarding L[n] came up when attempting to use it to derive the resonator's control frequency. The goal of this resonator is to generate the fundamental which appears in the spectrogram of figure 3. Clearly you would expect this to be related to the slide's velocity as that controls how frequently the spatially periodic pattern of windings would be encountered by the surface of the slide. However, how exactly this is achieved isn't specified in the CMJ article, nor Pupputi's thesis. The answer to how this is done lies in previous work by the same group of researchers on modeling the Chinese guqin. From it we get the following formulas [3]:

$$\omega_c = \alpha \frac{|\Delta f_0| * 2\pi}{F_s} \quad \text{and} \quad \Delta f_0 = f_0[n] - f_0[n-1]$$

$f_0[n]$ represents the fundamental frequency of the waveguide at time-step n and α is a scaling parameter tuned during development. The code from Puputti's thesis uses a value of α = 50,000, however no indication is given as to where this value comes from [4]. Beyond the lack of a proper technical explanation in the immediate literature, figure 2 (taken from the original CMJ paper) clearly indicates a different control signal is used. Granted, the expressions above can be generated by reverse engineering the signal labeled $f_c[n]$, however this would add unncessary work and also doesn't match the implementation by Pupputi in his masters thesis. Discovering this and properly tracking down an explanation was an unforeseen aspect which threw off the development cycle a bit.

Artifacts When Sliding

One of the other issues which came up during development was an audible distortion during the sliding process. The sliding process adjusts the total length of the digital waveguide in real-time which effectively performs a re-sampling on the audio data stored in the delay lines. Given this interpretation, an anti-aliasing filter was added in an attempt to mitigate the issue. It was able to reduce the magnitude of the artifact slightly, however not completely. Another approach which could be used is to bandlimit

the waveform that the digital waveguide is initialized with to see if that eliminates the issues more fully. Unfortunately due to time-constraints this wasn't investigated fully.

Further Development

There is clearly quite a bit of work which still remains to be done here. As mentioned in the previous section, fully debugging the artifacts which occur during the synthesized slides is a high priority. Developing the features which implement the contact sounds for unwound strings also remains to be done. Additionally, more advanced verification techniques could be used to track the fundamental frequency of the synthesized sound over time to ensure the synthesis algorithm and its control have been implemented correctly. An approach to this could be the YIN algorithm. One more fully developed and verified, some iconic slide licks could be implemented as a test of the musicality of the algorithm.

Source Code

The code used to generate the sound files shown in class can be found here:

https://github.com/dgsmith1988/Masters-Thesis/blob/main/Code/tests/test_StringSynth.m

The rest of the code regarding the system's implementation can be found here:

https://github.com/dgsmith1988/Masters-Thesis/tree/main/Code

Bibliography

[1] J. Pakarinen, T. Puputti, and V. Välimäki, “Virtual Slide Guitar,” *Comput. Music J.*, vol. 32, no. 3, pp. 42–54, 2008, doi: 10.1162/comj.2008.32.3.42.

[2] V. Välimäki and T. Tolonen, “Development and Calibration of a Guitar Synthesizer,” *J Audio Eng Soc*, vol. 46, no. 9, pp. 766–778, 1998.

[3] H. Penttinen, J. Pakarinen, V. Välimäki, M. Laurson, H. Li, and M. Leman, “Model-Based Sound Synthesis of the Guqin,” *J. Acoust. Soc. Am.*, vol. 120, no. 6, pp. 4052–4063, 2006, doi: 10.1121/1.2360422.

[4] T. Puputti, “Real-Time Implementation of the Virtual Air Slide Guitar,” Masters, Aalto University, 2012. [Online]. Available: https://aaltodoc.aalto.fi/handle/123456789/3258