Comparative Analysis of Physical and Virtual Spring Reverb Technologies

Shaojun Chen(s736chen), Jackson He(z262he), Honglin Cao(h45cao)

Friday 5th April, 2024

Abstract

This project aims to explore the auditory and technical differences between a physical spring reverb pedal, a virtual simulation of spring reverb implemented as a script and some commercially available spring reverb plugin. By designing, building, and testing a physical spring reverb pedal, and concurrently developing a digital emulation using Octave, we seek to understand how each medium affects audio signal processing and listener perception. The project will culminate in a comparative analysis based on technical measurements and subjective listening tests.

Contents

1 Introduction and Design

1.1 Background of spring reverb

Reverb, short for reverberation, is a fundamental audio effect that adds depth, spaciousness, and realism to sound recordings and live performances. It occurs when sound waves reflect off surfaces in an environment and then reach our ears, creating a complex blend of delayed and attenuated sound reflections. This phenomenon is what gives a sense of space, whether it's the natural reverberation in a cathedral, the intimate ambiance of a small room, or the ethereal quality of a dreamy musical track. Musicians, audio engineers, and producers often use various reverb techniques and equipment to enhance their audio recordings and achieve desired sonic characteristics.

Spring reverb was first introduced in the early 1930s by Laurens Hammond. Best known for the Hammond organ, was the first device to use a spring reverb unit. However, it was not until the 1960s that it became widely popular, thanks to its incorporation into guitar amplifiers.

Spring reverb have been a cornerstone in audio effects processing, providing distinctive reverberation effects that are difficult to replicate accurately with digital technology. The physical properties of springs contribute unique characteristics to the audio signal, which digital simulations strive to emulate. Spring reverb is known for its unique, sometimes "boingy" or "twangy" quality, and it has been used in various musical genres, from surf rock to dub reggae. It offers a different sonic character compared to other reverb types like plate, hall, or room reverbs, making it a favourite among musicians and producers looking for a vintage or distinctive sound.

1.2 Design of physical spring reverb unit

For this project, we firstly googled and find some photos of reverb tanks used in traditional guitar amplifiers. Most of the modern designs has the spring suspended in the gap of a transducer, and a transducer push and pulling on the spring causing them to vibrate at the same frequency as the audio signal.[1] And the receiving side was identical to the inputting side, with the vibration of the spring transformed into electronic signal by the transducer.

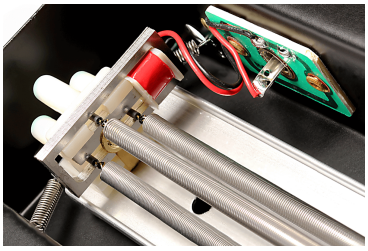


Figure 1: A closer look of a commercial reverb tanks

However, this type of spring reverb utilized custom-made transducer, which meant copying this design would be impossible for us, as we lacked the knowledge and tools to fabricate a custom transducer from scratch. So we went with an easier route: connecting

[1] "Mod Reverb Tanks - Spring Reverbs Explained." Mod Reverb Tanks - Spring Reverbs Explained — Mod Electronics. n.d. Web. 05 Apr. 2024.

the spring directly to a speaker, and let it pushing and pulling directly on the spring create longitudinal wave into the spring. This design might change the characteristics of the resulting signal, as the inputting wave onto the spring is longitudinal rather than transversal, and this could be an interesting comparison to look into.

For the speaker, we come across some very nice looking surface transducer, which is meant to transfer vibrations to large surfaces to create audible sounds, and they are essentially just conventional speakers with the paper cone replaced by a piece of metal.

For the receiving side, we decided to go with conventional guitar piezo under-saddle pickup to save some time on designing and manufacturing, plus we can use existing guitar amp or sound interface to get a cleaner sound from it rather than use some cheap amplifier board for a transducer.

As for the springs, since it's the core of any spring reverbs, we were unable to find any specifications for it used in commercial spring reverbs units. So we could only source some stainless steel pulling springs from ali-express and ems-store from Engineering 3 with various diameter and length, and we settled with one of them that have the lowest pulling force and largest diameter to hopefully achieving a longer decay rate, so the reverb effects could be more obvious.

1.3 Design of software spring reverb

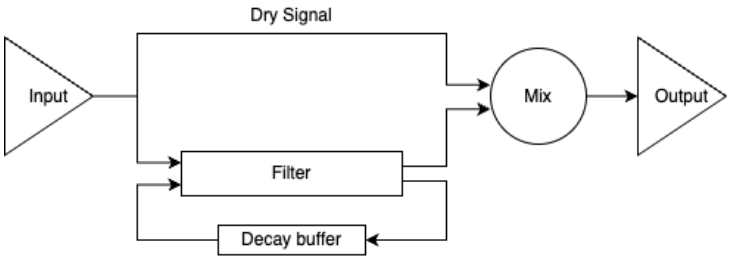
Initially we planned to develop a virtual version of a spring reverb as a VST audio plugin. However, we encountered some complexity issues of the algorithms, as we discovered the convolution processing is not possible to implement into a real-time plugin without dedicated hardwares, and all such plugins were heavily simplified and very difficult to understand. Therefore, we decided to use Octave for asynchronous audio processing. To achieve this, we carefully read 2 papers and summarized the principles of physical spring reverb into devising a relatively simple algorithm so we could finish in the timeframe of this term, since the primary goal was to model the physical spring with algorithms rather than making it pleasing to listen.

We referenced two pieces of articles and designed a simplified spring reverb model based on the content of "Efficient Dispersion Generation Structures for Spring Reverb Emulation" by Julian D. Parker. This paper focused primarily on enhancing the efficiency of digital simulation for spring reverb, as well as how to reduce the computational cost of digital reverb. The paper mentioned that traditional digital spring reverb's reliance on all-pass filters and delays to achieve dispersion effects, which was also an important reference part of our design. The main portion of the article discussed how to improve the computational efficiency and simulation fidelity of digital reverb by splitting different components based on traditional digital reverb. However, our goal was to simulate spring reverb using software, so we didn't need to consider efficiency issues a lot. Therefore, we only referenced the first half of the article.

From our experience in making the physical spring reverb, we knew that when sound passes through a spring, it experienced certain frequency losses due to the spring's inherent frequency, making the sound warm due to the reduction of high frequencies after passing through the spring. Therefore, we designed a low-pass filter to reduce the high frequencies of the signal. Moreover, a portion of the signal bounced back and sent to the output after the sound passes through the spring again, which was implemented with a delay line to simulate this physical property of the spring.

The fundamental framework included digital audio processing in Octave, filter design

and simulation of the propagation and the decay of waves inside the spring. The filter was to simulate the dispersion effect and frequency decay of the spring. The feedback delay network with decay was to simulate multiple reflections and propagation of signals in the spring. Finally, the wet signal will be mixed with the original signal and sent to the output.



In the above figure, we splited the input into two copies. One for the dry mix, and the another one for the spring algorithm to precess. Finally, we mixed them and wrote it as the output file. For the spring signal, the copy of the input first goes through a filter. Then, the processed signal was sent to the output, and a portion of the decay signal is sent back through the delay line into the filter again. The signal undergoes multiple overlaps, presenting a delayed effect. By controlling the delay time and adjusting the filter parameters, we achieve a simple simulation of the spring reverb effect.

1.3.1 Reverb model

The model we purposed had the following two main components:

- **Delay Network**: A set of delay buffers is used to simulate the multiple back-and-forth travels of sound within a spring. Each buffer contains the feedback signal for a specific time delay, which can easily record the delayed signal in different round trips. For each loop or round trip, the feedback level of the sound will decrease. Thus, we design a decay rate to adjust the feedback level after each sound reflection, simulating the energy loss as the sound propagates through the spring material.

- **Filter**: Due to the intrinsic frequency and material characteristics of the spring, sound signals passing through the spring will result in the decay of a certain frequency. Our filter, based on the spring's intrinsic frequency, makes corresponding adjustments to specific frequency bands of the sound, particularly the high frequencies. This approach renders the high frequency of the spring reverb softer.

And the following parameters we could change:

- **Feedback level & Decay rate**: These parameters together determine the duration and decay speed of the reverb. Higher feedback gain and lower decay rates result in a longer-lasting reverb effect, while the opposite leads to a quicker dissipation of reverb. To simulate the spring reverb. Adjusting these parameters can simulate springs of different material properties. We can modify the "`initial_delay_ms`" and "`decay_rate`" in the code to adjust the feedback level and decay rate.

- **Delay Time & Echo times**: The delay time determines the time in milliseconds that the signal takes to propagate within the spring. The echo times determine the

number of round trips a signal can travel. The longer delays can simulate longer springs, creating a richer reverb effect. The delay time is always set to be 20ms to 50ms. We can modify the "`initial_delay_me`" and "`num_echoes`" in the code to adjust the delay time and echo times.

- **Mix Rate**: The mix ratio of the reverberated signal and the original signal. The big mix rate will produce more spring reverb signals, while the opposite leads to a less reverb sound. We can modify the "`mix_rate`" in the code to adjust the mix rate.

- **Filter Frequency**: Adjusting the cut-off frequency of the filter can simulate the impact of spring physical characteristics on the sound's frequency response. A lower cut-off frequency simulates greater decay of high-frequency components, producing a warmer, more natural reverb effect.

2 Implementation

This chapter offers an in-depth examination of our journey from initial concept to actual implementation, covering the development of both hardware and software aspects of the spring reverb project. We also detail the methodologies employed in creating the testing clip and in deriving results from a commercially available reverb unit.

2.1 Generating testing clip

We recorded the testing audio clips using a Fender Telecaster and a Audient sound interface, with "Neural" plugin on Logic Pro as a guitar amplifier simulator. Three audio clip were recorded to have a comprehensive testing suite for the effect of a reverb unit. The input WAV files were `input_chord.wav`, `input_melody.wav` and `input_short_note.wav`.

Figure 2: Recording setup

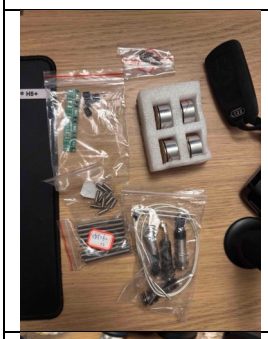
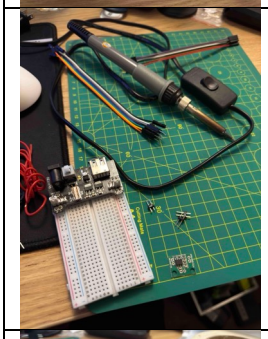
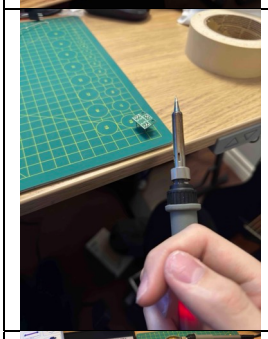
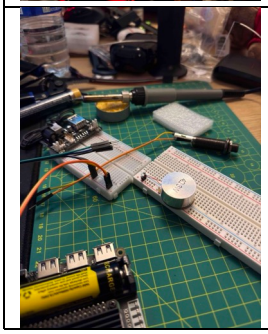
We chose to record internally rather than using microphone was because we could completely avoid any possible reverb effect from the physical environment we were in, so we could easily compare the effect later on.

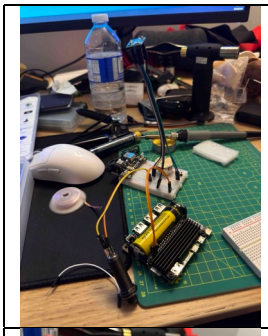
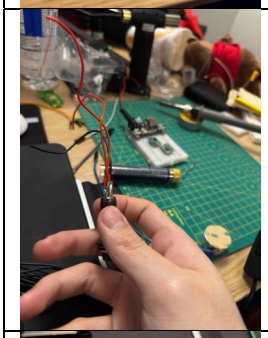
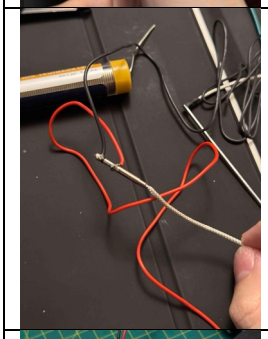
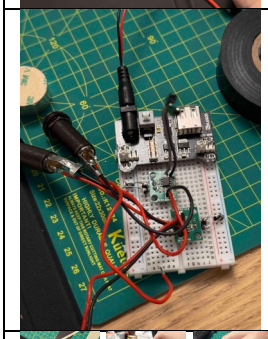
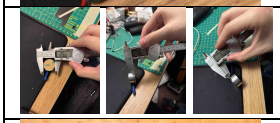
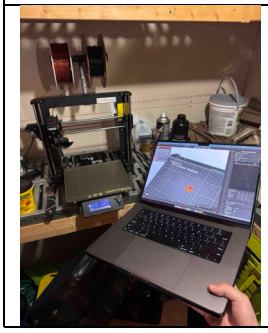
2.2 Generating reference clip

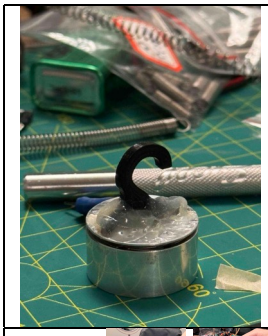
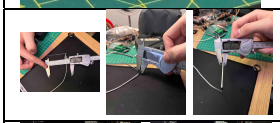
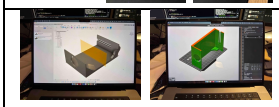
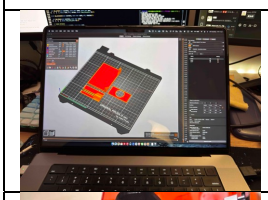
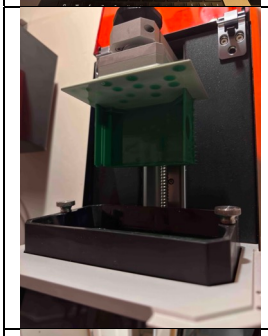
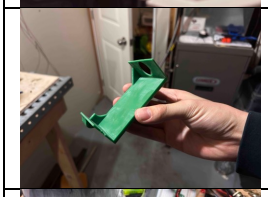
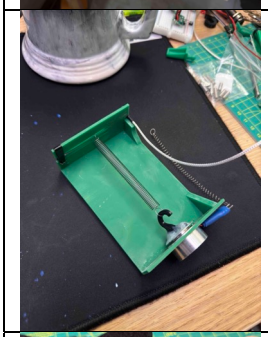
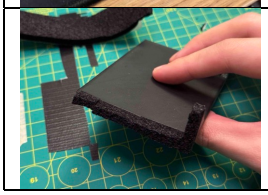
To provide a controlled reference for our evaluation, we applied a Logic built-in spring reverb effect plugin to the input files. The pre-set of the Logic Spring Reverb is: TIME = Medium, Tone = -20, STYLE = Boutique, MIX = 60%.

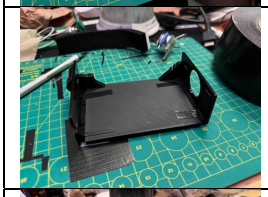
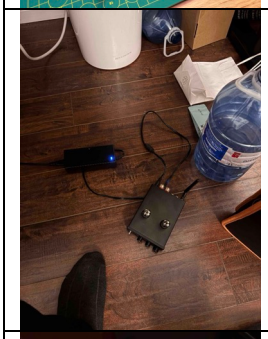
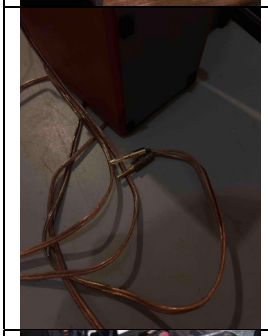
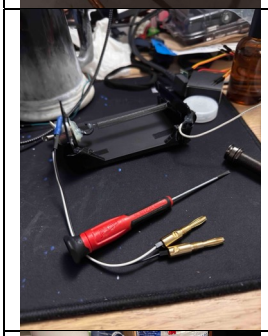
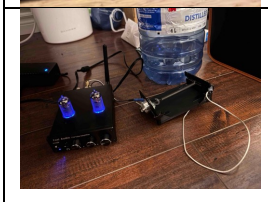
Figure 3: Reference clip setup

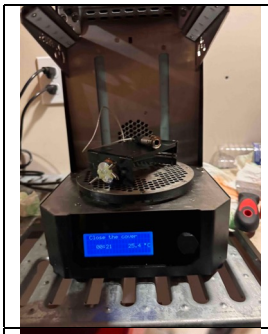
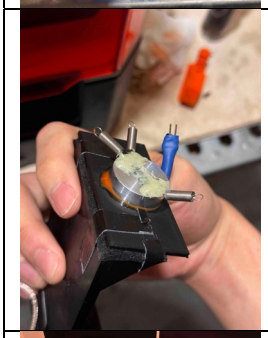
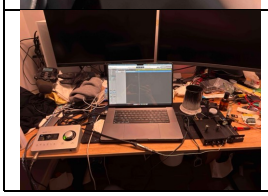
2.3 Hardware spring reverb

Photo	Description
	purchased surface transducer, driver board for the transducer, guitar piezo pickup, 6.5mm audio jack and several metal springs.
	Used a power supply board and a bread board from my Arduino kit, prepare to solder the wires.
	Soldered the pin to the transducer driver board.
	The power supply board needs 9V battery, trying to use my Raspberry PI Zero 2W to power the driver board instead. Used multimeter to determine the 5V and GND.
	Soldered the surface transducer to a GPIO pin for easier connection later.

	Connected both the driver board and the transducer itself to the bread board with Arduino cables.
	The 6.5 jack I purchased is stereo, even though we are only doing a mono output, but I soldered 3 wires to it.
	Soldered piezo pickup to 2 wires as well, i want to try to drive the piezo pickup with the transducer driver board.
	Redo the GPIO pins only for 5V and GND so I can mount driver boards directly onto the bread board, and soldered the 6.5 Jack to the input and output of driver boards respectively, also soldered the piezo pickup to the input of one of the driver board.
	Measuring the dimension of the surface transducer for designing housing for the spring reverb.
	Design and FDM 3D printed a hook that i will glue to the surface tranducer to hold the spring.

	Glued the hook to the surface transducer.
	Doing some measurement on the piezo pickup for designing housing for spring reverb.
	Designed the housing for spring reverb in Fusion 360, output to PrusSlicer for printing on SLA resin printer.
	Also designed a seperated version to glue on together later, just to make sure at least one version will be available.
	SLA resin print finished
	Post processing done
	Mount components to the housing, dimension seems correct.
	To prevent vibration of the surface transducer, applied automotive foam tape to the housing printed previously for gluing up.

	Applied foam tape to the two sides, so when bounding the 3 piece together the vibration wouldn't affect the piezo pickup system.
	Connect the 3 piece using duct tape, so the joint (foam tape) is flexible and able to absorb vibrations.
	Installed all the components to the housing.
	The amplifier for the surface transducer is broken somewhere, it's no longer functioning. Will use my bookshelf speaker amp to drive the surface transducer, it should be even better than the original amplifier I built for the surface transducer.
	Take the speaker cable connector from my DIY bookshelf speaker
	Connected the cable connector to the surface transducer.
	The amp works really well (it even has EQ adjustments), and the damping foam tape works as well, only around 20% of the surface transducer's vibration is able to reach the plate housing the piezo pickup. Now it shouldn't affecting the piezo pickup anymore.

	Some testing shows the surface transducer is moving inside the hole holding it creating some squeaking sound. Applied some UV curing resin to the surface transducer and using UV curing station for SLA printer to cure the resin.
	Now the surface transducer is permanently mounted to the housing plate.
	Final recording setup completed.

2.3.1 Recording setup

The driving of surface transducer was accomplished by connected an iPhone to the amplifier via BlueTooth, which connected to the surface transducer through two GPIO extension cable. The piezoelectric pickup which attached to the spring, was connected to my sound interface **Apollo Solo**, which in turn was connected to a laptop running**Logic Pro**. Then, we played the test clip from the iPhone and started recording in Logic Pro.

Due to the piezo pickup being designed for guitar use, its sensitivity was insufficient for capturing vibrations from the spring. Therefore, we had to increase the gain for the piezo pickup to +52 dB on the sound interface (hardware gain) to capture any sound, which also amplified the noise. Subsequently, we have to use the **X-noise** plugin from Waves to mitigate these noises, otherwise we cannot compare the spectrum later on. Even though applying the noise reduction plugin resulting in reduced detail.

Figure 4: Using plugin to cancel the noises

2.3.2 Future developments

During the recording process, we found that when the amplifier for the surface transducer was turned too high, it would resulting in the spring touching itself (too much vibration on the spring), which will create some rather loud artifact and disrupt the actual signal. So, we have to lower the gain of the amplifier, which resulted in needing to +52 dB on the receiving end while recording, which resulted in an extensive amount of noise.

Similar issue wasn't found on the commercial spring reverb tanks, presumably due to the fact that our implementation created longitudinal wave rather than transversal comparing to the commercial solutions; And it may also due to the fact that the spring we used has too many coil numbers, so it's very easy for the coils to touch each other when the amplitude was large. To mitigate this, we could find some extension spring that has the same diameter for the wire but way less coil number. However, the one we got was the only one left in the E3 mechanical store. So finding a suitable spring was quite hard and couldn't be completed in time for this project.

Secondly, even though the spring was not suitable, the piezo pickup was definitely not sensitive enough for capturing vibrations of the tiny spring. To improve this, we will need

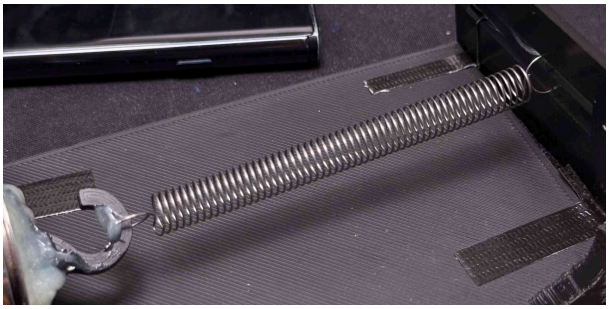


Figure 5: Spring used in my reverb tank

to find some other piezo pickup systems that are not designed for under-saddle usage for guitars, rather for sticking on the surface of an instrument to capture the vibrations, which should be more sensitive. Also, this couldn't be accomplished due to the time constraints of this project.

Lastly, the use of foam tape with duct tape seems quite effective for stopping the vibration, but after sitting there for a few days, it showed some level of deformation under the tension of the spring. So, some other mechanism not involving duct tape and foam tape was needed. we had originally decided to use some spring to suspend the surface transducer somewhere, but the supporting structure for this seemed not possible to be 3D printed. Also, to design and actually create such structure was also impossible to accomplish due to the time constraints of this project.

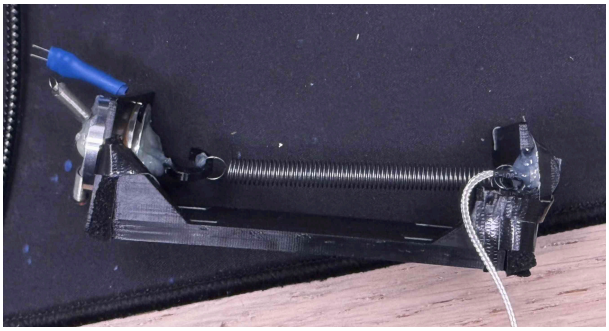


Figure 6: Housing is no longer straight under the tension of the spring

2.4 Software spring reverb

The implementation of the software Spring Reverb was in the Octave script "`spring_reverb.m`". This function will take an input WAV file and produce a processed output file in mono signal.

We can modify the file name in line 40: `[input_signal, fs] = audioread('test_melody.wav')` to modify the input file. Then, in last line 93: `audiowrite('output_melody.wav', reverbed_signal, fs)` you can modify the output file name. Also, to achieve a different spring effect, we can modify the parameters in the code as described in section 1.3.1.

To implement the filter, we used a Gaussian filter centred around the natural frequency of the spring. We applied FFT and IFFT to obtain the filtered signal. The `spring_filter` function was used to filter the input signal, simulating the effect of the spring's natural frequency on the sound. While implementing the delay network, we used a loop and buffer to record the decay signal to add the reverb effects to the output signal. We recorded the delay buffer and calculated the superposition of multiple echoes. The intensity of each was decreased according to the feedback gain and decay rate.

Finally, the reverb signal and the original signal were mixed following a set ratio (60% wet in our code[2]) and written as the new output signal.

Algorithm 1 Spring Reverb Simulation Pseudocode

Load the signal processing package.

procedure SPRING_FILTER($input\_signal, fs, f\_natural$)
    Perform FFT on $input\_signal$.
    Generate a Gaussian filter centered around $f\_natural$ in the frequency domain.
    Apply the generated filter to the FFT results.
    Perform IFFT to obtain the filtered signal.
    return $filtered\_input\_signal$
end procedure

Read WAV file into $input\_signal$.
Convert $input\_signal$ to mono if it is in stereo.
Initialize simulation parameters:

- N: Length of $input\_signal$
- $output\_signal$: An array of zeros with length N
- $initial\_delay\_ms$: The initial delay length in milliseconds
- num_echoes: The total number of echoes
- $feedback\_initial\_level$: The initial feedback gain level
- mix_rate: The mix ratio of the reverberated signal and the original signal
- $decay\_rate$: The decay rate after each reflection
- $spring\_natural\_freq$: The natural frequency of the spring

Initialize the delay buffer and calculate the delay sample length for each echo.
Preprocess $input\_signal$ with SPRING_FILTER function.
for each sample in $input\_signal$ **do**
    Add the current sample to the delay buffer.
    for each echo **do**
        Calculate the echo sample using the delay buffer.
        Accumulate the echo samples.
    end for
    Add the accumulated echo samples to $output\_signal$.
end for
Mix the original signal with $output\_signal$ based on mix_rate.
Normalize the mixed signal.
Write the normalized signal to 'output_melody.wav'.

[2]If we set the mix ratio to 100% wet, all the outputs will sounds just like a delay effect due to our implementation. Plus, our physical reverb implementation's housing also directly transmitted portion of the dry signal to the piezoelectric pickup. And 60% was decided because it produced the most pleasing tone on our testing clip.

2.4.1 Future developments

Our current model incorporated a basic implementation of dispersion through delay line and filtering. Moving forward, we will explore more advanced physical modelling techniques to simulate the dispersion effects. For instance, we can further calibrate the phase of the audio signal to achieve the effect of dispersion, and incorporate the impulse response (IR) of an actual spring to achieve a more complex and authentic reverb effect.

Although we omitted the issue of computational efficiency this time to simplify the process, we could investigate into turning the algorithm we proposed into a real-time software in the future, which enable it as a tool we could use for live performance and real-time recording applications. This involves refactoring our code architecture, improving overall execution efficiency and considering the delay in the output of the effect. We will reference the sections on algorithm efficiency in the paper "Efficient Dispersion Generation Structures for Spring Reverb Emulation" for improvements.

3 Evaluation and Comparison

3.1 Hearing comparison

Firstly, we were very surprised that our digital spring reverb generated almost exactly the same effect as the commercially available spring reverb in Logic Pro. The only discrepancy came from the mix rate. Even though we set the mix rate for both of them to be 60%, but the reverberation effect was way more prominent than our digital spring reverb. However, comparing to the physical reverb unit we built, the resulting sound from them could be described as applied many iterations of delay effect rather than an actual reverberation effect.

In contrast of both software spring reverb unit as mentioned above, our physical spring reverb units had heavily modified the sound signature of the testing clip. The reverberation effect from it was much more organic and continuous, combined with the warmth it added to the clip, made it sounded much like a Chamber reverb effect.

3.2 Going deeper

Before moving into comparison, we pre-processed all audios types (short note, chord, melody) of different source (original input, digital reverb output, physical reverb output, logic reverb output) into normalized audio such that they become easier to compare.[3]

In the graphs below, we ordered the audio by

1. input audio
2. digital reverb output audio
3. physical reverb output audio
4. logic reverb (commercial) output audio

We will begin by examining the waveform of each audio type.

[3]Code given in Appendix

Considering the waveform of the short note audio, we saw that all three reverbs did a great job creating the reverberation effect of spring reverb, where they made the sound extending longer than the original input and decayed over time. However, we noticed that both digital reverb and logic reverb produced the short note with a strong instant entry, while the physical reverb produced the short note with a soft gradual entry. This behavior of the physical reverb is expected due to its fundamental property. The different behavior shown in the digital reverb and the logic reverb weren't expected, but can be explained to be the drawback of the algorithm used, or simply due to the lower mix rate being used such that more of the original input audio got mixed.

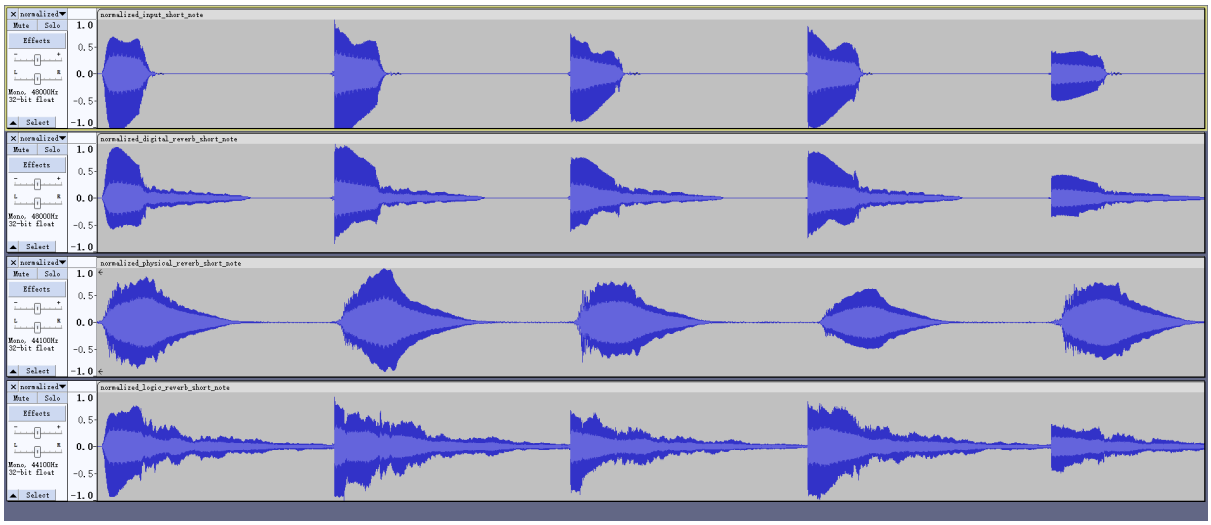


Figure 7: Waveform comparison for short notes

In the waveform of chords and melody, we saw the same feature as we discussed above, which the processed audio extended longer than the original input and decayed over time. However, we noticed that in the waveform of chords, the beginning of the physical reverb output has been slightly cut off. This was because the physical reverbs uses the output of WeChat as input, and WeChat automatically adds a fade in effect at the beginning of the audio, thus should not be a concern. We also noticed that the output of physical reverb tend to be softer and smoother than other source. This can be explained to be the fundamental property of a true spring reverb, and also due to the the lower mix rate being used in digital reverb and logic reverb such that more of the original input audio gets mixed.

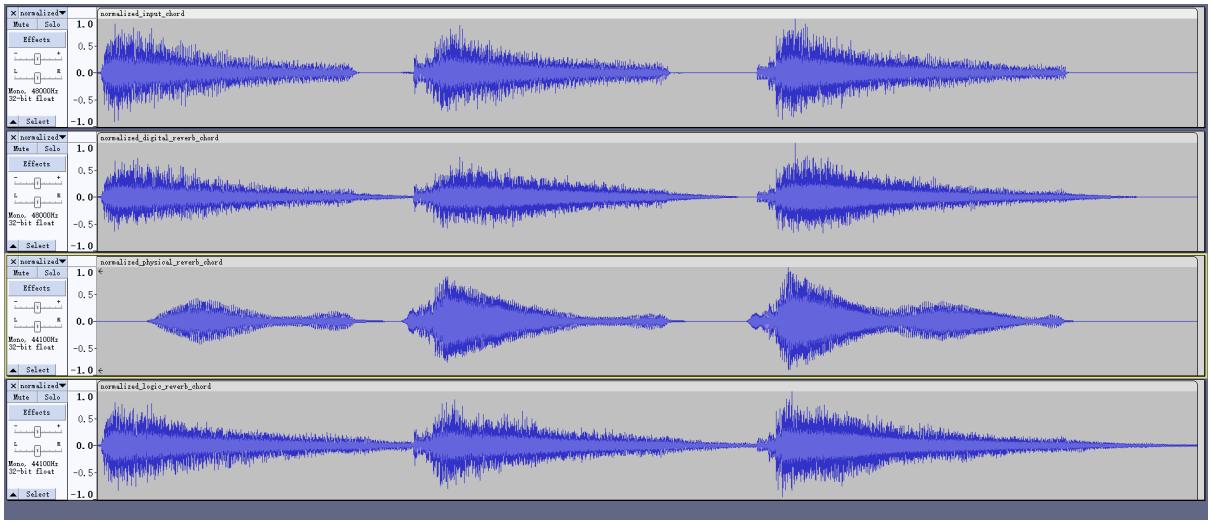


Figure 8: Waveform comparison for chord

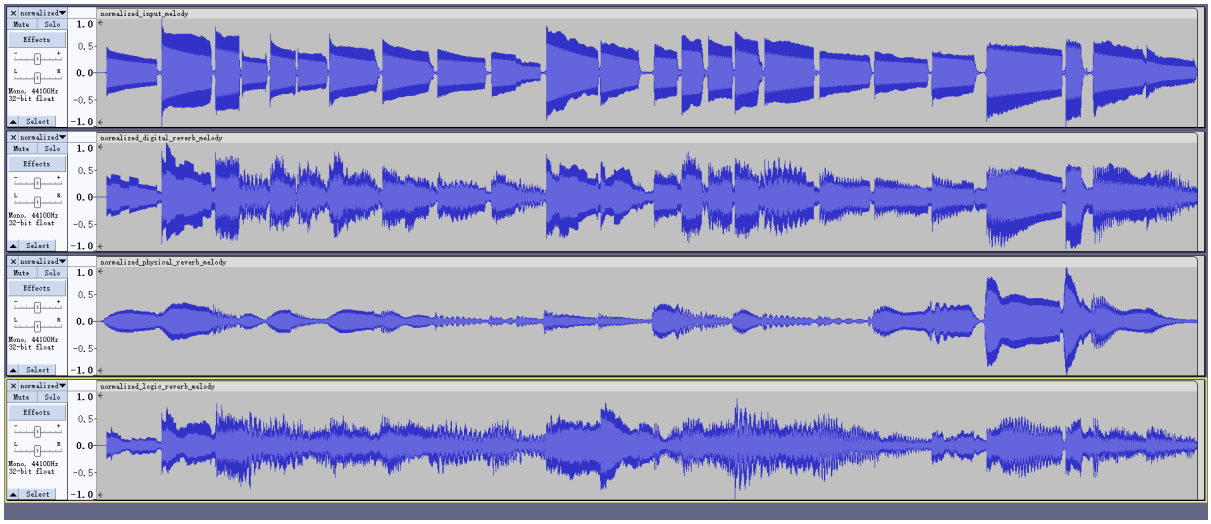


Figure 9: Waveform comparison for melody

In the spectrogram of the short note audio, we saw that the physical reverb filtered out a lot of the high frequencies and low frequencies in the audio (especially high frequencies). This conformed with our theory that the spring has a fundamental frequency, which frequencies close to its fundamental frequency would be kept and frequencies far away from its fundamental frequency would be filtered out. Our digital reverb also has the same property, but our filter was not aggressive enough and kept more higher frequencies compared to the physical reverb. This could be improved by changing the filter parameter and increase the mix rate.

In the spectrogram of both short note and chord audio, we also saw that lower frequencies in the physical reverb's output tend to last longer compare to higher frequencies. Our explanation for this was the higher frequencies tend to lose its energy faster when travelling through the spring. However, we didn't see such property in the digital reverb and logic reverb's output, where all frequencies last for about the same amount of time.

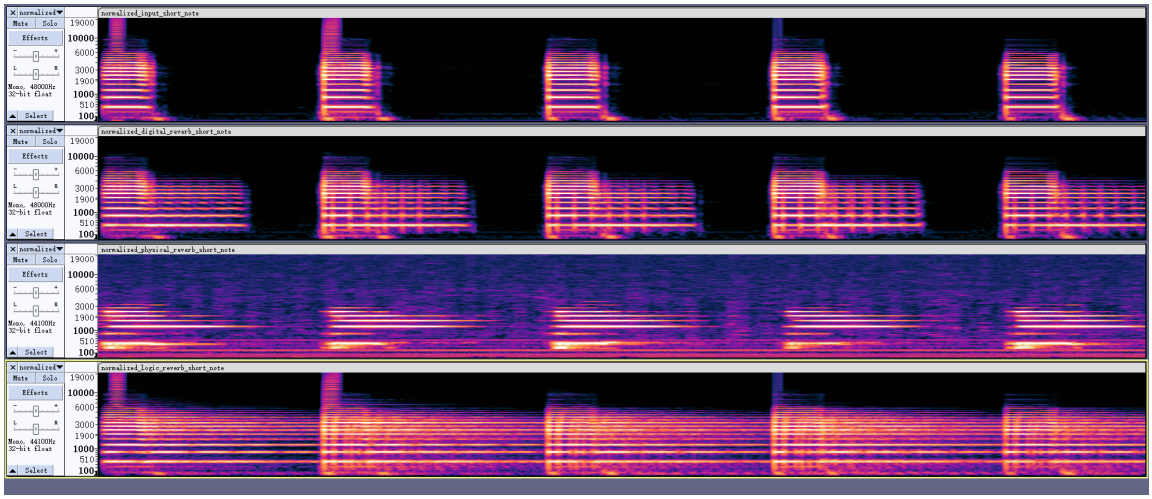


Figure 10: Spectrogram comparison for short notes

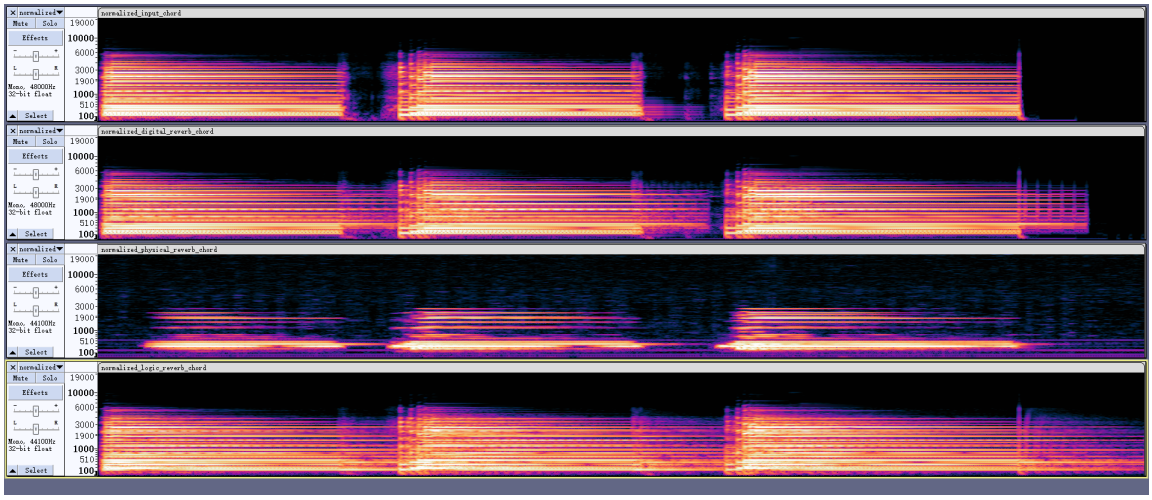


Figure 11: Spectrogram comparison for chord

Last, in the spectrogram of the melody, we saw some interesting results.

By comparing the input audio, digital reverb's output and the logic reverb's output, we see that more vertical bars appear in the spectrogram for the digital reverb and logic reverb's output. Our explanation for this was that, those were the result of add old signals remain in the spring to newer signals (that's our understanding of how the spring reverb works). We also saw that the logic reverb had the feedback added more frequently than the digital reverb.

Observing that the physical reverb didn't have such property appearing, we concluded that this might be because of the real spring reverb does the feedback process in a continuous time, where the software does the feedback process as a discrete computation.

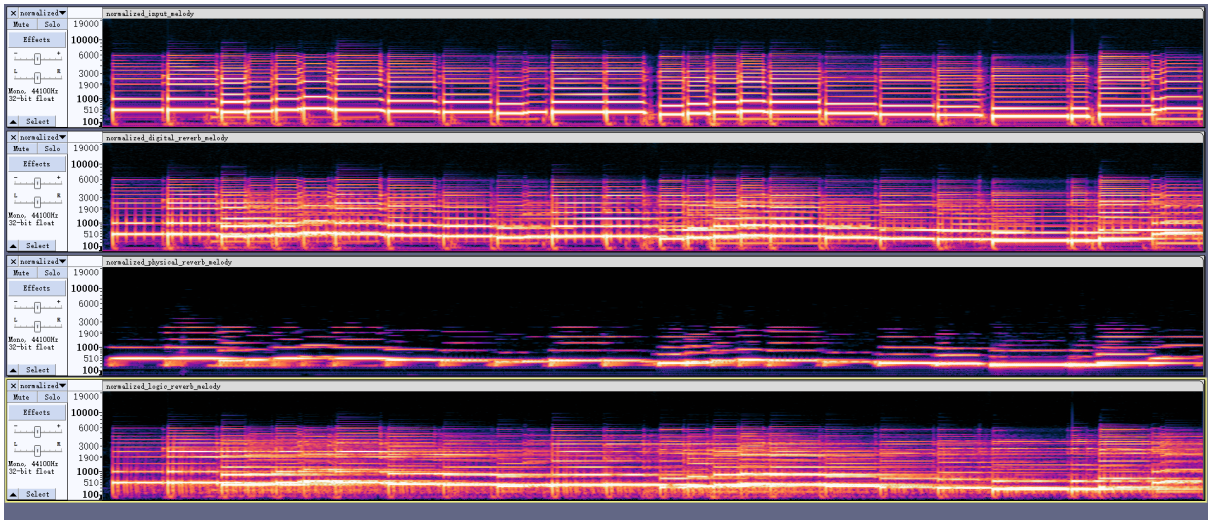


Figure 12: Spectrogram comparison for melody

4 Conclusion

Our efforts in this project were channeled towards the detailed investigation of the physical and virtual aspects of spring reverb, with the aim of finding links between them. By following a meticulous construction and analysis methodology of building an actual pedal based on spring reverb followed by its digital counterpart in Octave, we tried to grasp its springy and unique sound. Our comparative research, based on our work against commercial plug-ins, was yet another path that led us to discover how different media can affect signal processing and, thus, what a listener hears.

According to our study, the medium has a very strong influence on the reverb characteristics. The physical spring reverb has its own particular character; It is rich, warm, and constant - features that digital simulations still cannot match with great accuracy. The appeal of the analogue audio processing devices was linked to this warmth and organic quality born from interactions among different components of the spring. In contrast, the digital emulation was accurate and flexible; It highlighted the difficulties of reproducing the myriad colours, luminosity changes, and touch response in a physical instrument. Nonetheless, it exemplified well the adaptability and reach of digital technology, providing an equally satisfactory but much cheaper sustainable option that can keep improving in terms of quality.

The commercial plugins offered a balanced compromise, encapsulating a significant portion of the analog device's character while benefiting from the digital domain's flexibility. Yet, our direct comparison revealed the nuanced differences in sound quality and character between our project outputs and established commercial solutions, shedding light on the intricate dance between authenticity and practicality in sound engineering.

From our journey into the spring reverb world, we were able to see not only how it is put together technically and its characteristics sound-wise, but we also discovered a new area of cross-pollination potential between analog and digital realms. Given that there's more to these intersections for us to explore, understanding how audio signal processing works becomes much easier, opening up the horizons of sounds that are within an artist's reach, as well as for engineers.

A Code of digital spring reverb

```
pkg load signal

function filtered_input_signal = spring_filter(input_signal, fs, f_natural)
  %% taking FFT
  % Perform FFT on the audio signal
  sigma = 1;             % Standard deviation for the bell shape
  Y = fft(input_signal);

  % Generate frequency axis for FFT
  nfft = length(input_signal);
  f_axis = linspace(0, fs, nfft);

  % Convert frequencies to log scale
  log_f_axis = log2(f_axis);

  % Create the filter
  filter = exp(-((log_f_axis - log2(f_natural)).^2) / (2*sigma^2));
  filter = filter / max(filter); % Normalize the filter

  % Determine if nfft is odd or even
  if mod(nfft, 2) == 0
      % nfft is even
      midpoint = nfft / 2;
      filter = [filter(1:midpoint), fliplr(filter(1:midpoint))];
  else
      % nfft is odd
      midpoint = (nfft + 1) / 2;
      filter = [filter(1:midpoint), fliplr(filter(2:midpoint))];
  end

  % Apply the filter to the FFT of the audio signal
  for i = 1:length(Y);
      Y_filtered(i) = Y(i) * filter(i);
  end
  % Perform inverse FFT to get the filtered audio signal
  filtered_input_signal = ifft(Y_filtered);
end

% read file
[input_signal, fs] = audioread('test_melody.wav');

% make the file mono if it is stereo
if size(input_signal, 2) > 1
    input_signal = input_signal(:,1); % use the first channel
end

% Parameters
N = length(input_signal);
output_signal = zeros(N, 1);
initial_delay_ms = 40; % The initial delay length
num_echoes = 6; % Number of echoes
feedback_initial_level = 0.6; % Initial feedback gain
mix_rate = 0.6; % Mix ratio of the reverberated signal and the original signal
decay_rate = 0.8; % Decay rate after each reflection
spring_natural_freq = 1000; % Natural frequency of the spring

% Initialize the delay buffer
max_delay_samples = round(fs * (initial_delay_ms / 1000) * (1 + 2*num_echoes));
delay_buffers = zeros(max_delay_samples, 1);

% Calculate the delay sample length for each echo
delay_lengths = round(fs * initial_delay_ms / 1000) * (1:2:2*num_echoes);

filtered_input_signal = spring_filter(input_signal, fs, spring_natural_freq);

for n = 1:N
    current_sample = filtered_input_signal(n);
    echo_sample = 0;

    % Add new samples to the delay buffer
    delay_buffers = [current_sample; delay_buffers(1:end-1)];

    for echo_num = 1:num_echoes
        echo_index = delay_lengths(echo_num);
        % Calculate the echo level for this round
        feedback_level = feedback_initial_level * decay_rate^(echo_num-1);
        % Calculate the feedback
        echo_sample = echo_sample + feedback_level * delay_buffers(echo_index, 1);
    end

    % Add the echo samples to the output signal
    output_signal(n) = echo_sample;
end

% Mix the original signal and the reverb signal
reverbed_signal = (1 - mix_rate) * input_signal + mix_rate * output_signal;

% normalize
reverbed_signal = reverbed_signal / max(reverbed_signal);

% Output
audiowrite('output_melody.wav', reverbed_signal, fs);
```

B Code to normalize sound clips

```
pkg load signal;

function normalize_and_write_audio(input_file, output_file_name)
    % Read audio file
    [y, fs] = audioread(input_file);

    % Normalize audio amplitude to range [-1, 1]
    max_amp = max(abs(y));
    y_normalized = y / max_amp;

    % Write normalized audio back to file
    audiowrite(output_file_name, y_normalized, fs);
end

function normalize_all_wav_files_in_folder()
    % Get a list of all .wav files in the folder
    folder_path = pwd();
    wav_files = dir(fullfile(folder_path, '*.wav'));

    % Loop through each .wav file and normalize
    for i = 1:length(wav_files)
        file_name = fullfile(folder_path, wav_files(i).name);
        output_file_name = fullfile(folder_path, ['normalized_' wav_files(i).name]);
        normalize_and_write_audio(file_name, output_file_name);
        disp(['Normalized and wrote ' file_name]);
    end
end

normalize_all_wav_files_in_folder();
```