FAUST+JUCE Implementation of Leslie Speaker Simulation

Riccardo Miccini
Department of Architecture, Design and Media Technology,
Faculty of IT and Design,
Aalborg University Copenhagen
rmicci18@student.aau.dk

***Abstract*—This document presents a virtual simulation of the Leslie speaker commonly found in Hammond electric organs. The digital effect has been developed using the FAUST programming language, and then ported into a VST-compatible format using the JUCE C++ framework.**

Firstly, an overview of the perceptually relevant phenomena occurring in the speaker is provided. Subsequently, each isolated phenomenon will be generalized using suitable models, with focus on its software implementation.

The resulting effect is then validated in terms of its fidelity, performances, and overall pleasantness. Conclusions will then be drawn, highlighting the challenges, learning outcomes, and possible future improvements.

I. Introduction

The Leslie speaker is a combination of amplifier, loudspeaker, and audio effect dating back to the 1940s; originally intended to be used on electric organs, its popularity cause it to be adopted by a variety of performers and sound engineers. It is composed of a wooden cabinet divided into separate compartments along the horizontal plane. The bottom compartment holds a rotating baffle with an open side; a woofer is mounted over the drum, facing downwards, allowing the sound to radiate from the rotating opening. The upper compartment holds a rotating platform hosting a compression driver and an acoustic horn facing outwards.

The input signal is amplified by a vacuum tube amplifier and then split into the two drivers by a 800 Hz passive crossover. Each rotating part is driven by a separate motor at roughly the same speed, with two settings available .

The sound emitted by the rotating horn and woofer undergoes a complex series of transformations resulting in a dense and rich effect, similar to a chorus.

II. Sound characteristics

Given the complex nature of the apparatus, it is important to discern which perceptual cues are significant. The work by [1] determines that three perceptual attributes are altered: space, timbre, and pitch. The following subsections aim at highlighting the specific effects on each attribute, and are based on the research of [2], [3].

A. Timbre variation

Variations in spectral content are mainly caused by the following three elements:

- **Tube amplifier**: the vacuum tube amplifier equipped in some Leslie models confers a warm tone to the original signal
- **Horn**: the acoustic horn used for radiating the sound has a characteristic frequency response consisting of a shallow peak at approximately 2 KHz, as documented in [4]
- **Air absorption**: the propagation of sound waves though air attenuates high-frequency components. This is particularly noticeable due to the source being at different distances to the listener over time

B. Space

When captured from two microphones placed at equal distance from the source, the spinning nature of the sound source becomes obvious; this is caused by the high directionality of the horn, and as such is mostly audible in the trebles. The exact location of the source is not intelligible from the woofer signal alone, with the rotation of its baffle only affecting its loudness.

C. Frequency shift

An effect similar to a vibrato can be heard at high enough rotation velocities. This is caused by the Doppler effect manifesting itself when there is a relative velocity between the source and the receiver (and vice versa). At any given time, the relative velocity seen by either microphone is different, causing a different amount of shift which contributes to the rich tone of the effect.

III. Models and implementation

This section documents the models used to recreate the characteristics described above, as well as their implementation. The techniques used here are derived from [5], [6].

A. FAUST

FAUST is a domain-specific programming language used for the development of signal processing algorithms. It follows a purely functional paradigm, and as such it is completely depleted of mutable data or internal states.

Data, in the form of signal streams, is processed using functions, adhering to their strict mathematical definition.

Functions can be routed together using composition operations, in a way that is similar to block diagrams.

Code written in FAUST can be compiled or exported to a variety of platforms and devices, or directly executed using its run-time environment.

B. Tube amplifier

The response of the vacuum tube amplifier can be modeled using the following wave-shaper equation[6]:

$$y(n) = \frac{\arctan(k \cdot x(n))}{\arctan(k)}$$

where k is the gain factor. The maximum value of k is limited by the amount of tolerable aliasing.

C. Horn response

The frequency response of the acoustic horn mentioned above has been rendered with the peaking equalizer filter available in FAUST. The filter has been configured to have a gain of 10 dB and a bandwidth of 4 KHz, while the center frequency is user-selectable with a default value of 2 KHz.

D. Rotating platform

The angle of the rotating apparatus can be easily described by a sawtooth wave with amplitude in the range $[0, 2\pi]$; this signal will be referred to as $\Theta(t)$.

For a given angle θ, the distance between the sound source (i.e. the mouth of the horn) and the receiver has been derived to be:

$$d(r, \theta) = \sqrt{(r + r\cos(\theta))^2 + (r + r\sin(\theta))^2}$$

where r is the radius formed by the center of rotation and the source (i.e. the length of the horn).

The two equations above can be then combined to obtain the distance between source and receiver over time:

$$D(r, t) = d(r, \Theta(t))$$

Due to the trivial and predictable nature of $\Theta(t)$, the two distance equations will be used interchangeably hereafter.

E. Amplitude modulation

The change in loudness caused by the moving source can be modeled using AM (Amplitude Modulation). The waveform used to modulate amplitude is the previously described $d(r, \theta)$ function, inverted and normalized in the $[-1, 1]$ range:

$$d'(\theta) = \sqrt{(2)} - \frac{d(r, \theta)}{r}$$

Moreover, in order to operate on the perceived loudness of the signal rather than its amplitude, the following mapping is applied:

$$A_{dB}(\theta) = 10^{\frac{d'(\theta) \cdot k}{20}}$$

where k controls the depth of the modulation.

F. Modulated LP filter

The timbrical changes caused by air absorption have been modeled as a 2-order Butterworth low-pass filter with moving center frequency. The center frequency varies according to the following equation:

$$f_c(\theta) = f_b \cdot 2^{d'(\theta) \cdot k}$$

where k controls the depth of the modulation, f_b is the base frequency, and $d'(\theta)$ is the inverted and normalized distance function.

G. Doppler effect

The doppler shift for a moving source and still receiver is simulated using the delay lines method presented in [7]; the read-pointer is incremented by $1 + v_{ls}/c$, where v_{ls} is the velocity of the listener relative to the source and c is the speed of sound.

The FAUST implementation is based on the `rwtable()` primitive, which provides a read-write table for storing a portion of a signal. At each internal iteration, the write pointer is incremented by 1 and the read pointer is incremented according to:

$$1 - \frac{D(r, \theta)}{c} SR$$

where SR is the sampling rate. Running the FAUST code in the online editor shows the position of the read and write pointers along a bar graph.

H. Signal routing

The original signal is initially processed by the wave-shaper, and then fed into a variable-frequency crossover, modeled using two cascaded 2-order Butterworth low-pass and high-pass filters for each signal split respectively.

The bass path is split again at 200 Hz; the higher end is then processed with the AM block and summed back to the lower end which stays unaffected. The bass uses its own instance of the $\theta(t)$ block, allowing for independent speed selections.

The treble path is then split into two channels, representing the left and right microphones. Each channel $c(t)$ receives the following processing:

$$c(t) \rightarrow \text{doppler}(r, \theta) \rightarrow \text{am}(\theta) \rightarrow \text{lpf}(\theta) \rightarrow \text{horn}()$$

where θ is the instantaneous angle described by the rotating horn, and has a 90-degree offset between the two channels. Each treble channel is then separately mixed with the bass path and, resulting in a stereo signal.

I. Plugin and UI

While FAUST provides basic UI elements, the user interface for the final implementation has been developed using the JUCE, a C++ framework targeting various audio plug-in formats such as VST.

In order for the FAUST code to run within the C++ environment, it had to be exported using the `faust2api` command-line tool and the resulting class instantiated within

the JUCE `PluginProcessor` class, which operates as a controller.

The interface features the following controllable parameters:

- **General** → **Amp drive**
- **General** → **Crossover freq.** (Hz)
- **Bass** → **Rotation speed** (RPM)
- **Bass** → **AM depth**
- **Bass** → **Mix** (%)
- **Treble** → **Rotation speed** (RPM)
- **Treble** → **AM depth**
- **Treble** → **Mix** (%)
- **Treble** → **Mics distance** (deg.)
- **Treble** → **LPF Mod.** → **Center freq.** (Hz)
- **Treble** → **LPF Mod.** → **Depth** (oct.)
- **Treble** → **Horn** → **Radius** (cm)
- **Treble** → **Horn** → **Resonance freq.** (Hz)

Each control has a viable range of values, with custom interval and skewness factors. Moreover, they are band-limited in order to avoid clipping on value change.

IV. Results

The implementation documented above does an adequate job at approximating the complex range of phenomena affecting the sound of a Leslie speaker. However, it suffers from several major issues.

The `rwtable()` primitive only supports integer pointer values, and as such it jumps from one sample to another thereby generating unpleasant clicking noises. Moreover, the wave-shaper function used to simulate the amplifier introduces equally unpleasant aliasing components. Nevertheless, it is worth noting how both phenomena are mitigated by the low-pass filtering occurring at later stages of the processing chain; in fact, tweaking the control parameters (e.g. reducing the radius) effectively masks the problem.

Evaluating the fidelity of the final result proves difficult due to the complex nature of the original effect. However, upon several listening tests and comparisons with a variety of source materials and parameter values, it becomes apparent how the final result sounds *thinner* and *less dense*, lacking some of its original warmth.

Conversely, the possibility of individually tweaking the simulation parameters allowed for experimentation with setups that would otherwise be cumbersome or impossible to obtain with the physical device, such as different speeds for each driver, longer horns, and so forth.

Finally, when evaluating the performances in terms of computer resources used, the simulation behaves quite well, barely burdening the CPU and running comfortably in a web browser. This is due to FAUST's highly optimized DSP engine; of particular interest is how despite having declared two individual `rwtable()` instances (oen for each microphone channel, as the primitive does not support multiple read pointers), the FAUST compiler was capable of optimizing it into a single buffer in the resulting C++ code.

V. Conclusions

Using the existing literature and the FAUST and JUCE software tools, a perceptual simulation of the Leslie speaker has been conceptualized and implemented.

Despite the aforementioned shortcomings, the current implementation offers a decent variety of pleasant sounds, as well as solid starting point for exploring more sophisticated models.

Due to its fast-paced development, FAUST's documentation is inconsistent in regards to coverage of its built-in functions; code examples are fairly scarce and rarely compatible with the latest versions. Moreover, the lack of normally ubiquitous programming mechanisms such as arrays and stateful variables posed a challenge when formalizing the algorithms and adopting concepts from the literature, which is mostly tailored towards the imperative paradigm.

Nevertheless, the intuitiveness of its signal routing features and built-in support for a wide range of platforms makes it an interesting candidate for DSP-related projects like this one.

The simulation could certainly benefit from several improvements, in particular a fractional delay line with interpolation, as well as a simulation of the characteristics of the wooden cabinet, such as internal reflections and absorption.

References

[1] V. Verfaille and C. Guastavino, "An interdisciplinary approach to audio effect classification," in *Proceedings of the 9th International Conference on Digital Audio Effects, DAFx 2006*, 2006.

[2] R. Kronland-Martinet and Voinier, "Real-time perceptual simulation of moving sources: Application to the leslie cabinet and 3d sound immersion," *EURASIP Journal on Audio, Speech, and Music Processing*, 2008. [Online]. Available: https://doi.org/10.1155/2008/849696

[3] J. Pekonen, T. Pihlajamki, and V. Vlimki, "Computationally efficient hammond organ synthesis," in *Proceedings of the 14th International Conference on Digital Audio Effects, DAFx 2011*, 2011.

[4] C. A. Henricksen, "Unearthing the mysteries of the leslie cabinet." [Online]. Available: http://www.theatreorgans.com/hammond/faq/mystery/mystery.htm

[5] R. Penniman, "Rotary circus: A leslie speaker simulator," 2013, submitted to the 2013 AES Student Design Competition. [Online]. Available: http://www.rosspenniman.com/Rotary%20Circus%20-%20Description.pdf?attredirects=0

[6] W. Pirkle, *Designing Audio Effect Plug-Ins in C++*. Taylor & Francis, 2012.

[7] J. O. Smith, "Physical audio signal processing," 2010. [Online]. Available: https://ccrma.stanford.edu/~jos/pasp/