ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA Y SISTEMAS DE TELECOMUNICACIÓN

PROYECTO FIN DE GRADO

TÍTULO: Diseño e implementación de un instrumento virtual de Harpejji mediante síntesis por modelado físico

AUTOR: Alberto Barrera Herrero

TITULACIÓN: Grado en Ingeniería de Sonido e Imagen

TUTOR: Lino García Morales

DEPARTAMENTO: Ingeniería Audiovisual y Comunicaciones

VºBº

Miembros del Tribunal Calificador:

PRESIDENTE: Ignacio Antón Hernández

TUTOR: Lino García Morales

SECRETARIO: Antonio Mínguez Olivares

Fecha de lectura:

Calificación:

El Secretario,

Summary

Design and implementation of a virtual Harpejji instrument through synthesis by physical modeling

The Harpejji is a relatively new instrument that combines guitar-like construction with more piano-like playing. In this project, a polyphonic Harpejji synthesizer is developed using physical modeling synthesis to model the vibration of the instrument's strings, which is a type of instrument that does not exist on the market yet.

The result is a virtual instrument plug-in in VST3 format that is compatible with most current digital audio workstations (DAWs), both on Windows and MacOS systems. The plugin receives MIDI messages from the host DAW and plays to its output the sound produced by the combination of the vibration of all the virtual strings of the instrument. For its development, the free code library JUCE has been used and as programming language, C++.

The approximation of the wave equation by means of finite differences has been used as a physical modeling method to model the vibration of the instrument's strings. It is not the most efficient method, but it is relatively easy to implement, and good performance has been achieved with a sufficient number of voices for most applications.

The results of the project are considered to be satisfactory, although there is room for improvement. All the established objectives and design restrictions have been met and some improvement proposals are proposed that would make the instrument a really useful tool for use in musical production fields.

Abstract

Design and implementation of a Harpejji virtual instrument using physical modeling synthesis

The Harpejji is a relatively newly created instrument that combines a guitar-like construction with a more piano-like execution. In this project, a polyphonic Harpejji synthesizer is developed using physical modeling synthesis to model the vibration of the instrument's strings, which is a kind of instrument that does not yet exist on the market.

The result of the project is a virtual instrument plugin in VST3 format compatible with most current digital audio workstations (DAW), both on Windows and MacOS systems. The plugin receives MIDI messages from the host DAW and plays at its output the sound produced by the combination of the vibration of all the virtual strings of the instrument. For its development, the JUCE free code library and C++ programming language has been used.

The finite difference approximation of the wave equation has been used as the method for the physical modeling of the vibration of the instrument strings. It is not the most efficient method, but it is relatively simple to implement, and good performance has been achieved with enough voices for most applications.

The results of the project are considered satisfactory, although there is room for improvement. All the established objectives and design restrictions have been met and some improvements are proposed that would make the instrument a very useful tool for music production.

Content

list of acronyms

CPU	Central processing unit; Central Processing Unit.
DAW	digital audio workstation; Digital Audio Workstation.
DSP	digital signal processing; Digital Signal Processing.
IIR	Infinite impulse response; Infinite Impulse Response.
MIDI	Musical instrument digital interface; Musical Instrument Digital Interface.
VST	Virtual Studio Technology; Virtual Studio Technology.
VST3	VST Version 3.
VSTfx	VST audio effect.
VSTi	VST instrument.

Definitions

α	Level detector parameter. Inverse of the number of samples you use.
μ	Linear density of the chord.
ρ	Volumetric density.
σ_0	Linear dimming constant.
σ_1	Frequency dependent damping constant.
ω	Angular frequency of vibration.
ω_n	Angular frequency of vibration of the eigennumber mode.n
a	chord radius.
b_n	Amplitude of the vibration of the eigennumber mode.n
c	Velocity of propagation of the vibration in the rope.
c_n^2	RMS level detector value.
d	Distance between the nut of the instrument and the number fret.n
E	Young's modulus or longitudinal modulus of elasticity of the rope.
f_s	Sampling rate.
k	Wave number.
K	Turning radius.
L	String length.
R	Reflection coefficient.
s	Scale length. Distance between the nut and the bridge of the instrument.
S	Cross sectional area of the rope.
T	Tension in the rope.
T_{60}	Decay time. Time it takes for the sound level to drop 60 dB.
v_0	Initial velocity on the rope.

X	Chord length in number of points.
Z	Impedance at the end of the string.
Z_0	Characteristic impedance of the medium.

1.Introduction

An introduction to the Harpejji is made in this chapter, which includes a brief history of the instrument and a detailed review of its features. The different synthesis methods are also explained, and synthesis by physical modeling in more detail, as well as its advantages and disadvantages and why it has been chosen. Finally, the design restrictions or objectives that were established for the project in the draft phase are included.

1.1.The Harpejji

The Harpejji is a stringed musical instrument invented by Tim Meeks in 2007 and manufactured by Marcodi Musical Products, according to [1]. It combines characteristics of the guitar with a play and distribution of notes similar to that of the piano. It has a construction in different types of wood (birch, maple or bamboo), forming a surface on which the frets are placed. The execution is done by pressing the string with the fingers, as if the keys of a piano were pressed. The method of interpretation of the Harpejji is observed in theFigure1.

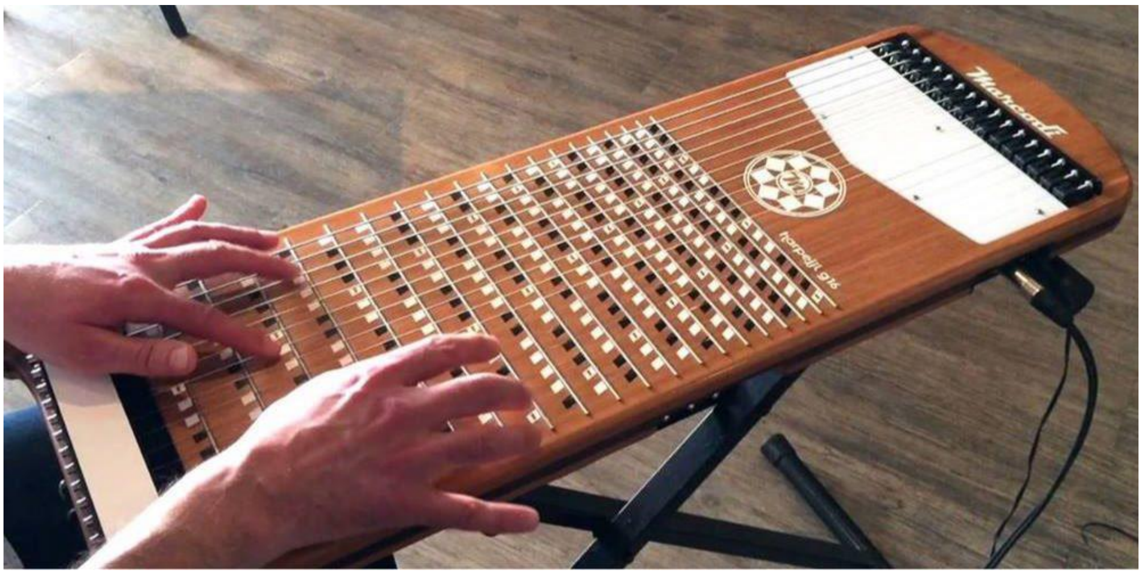


Figure1. Performer playing the Harpejji.

The frets are distributed, like those of a guitar, in semitones and between one string and the next there is a difference of one tone. This layout generates an isomorphic keyboard, which allows both chords and scales played in a given key to have the same layout when transposed to another.

At the time of the development of the project, there are three models, sold on the manufacturer's page [2], of three, four and five octaves, with 12, 16 and 24 strings respectively. The vibration of the strings is picked up by a piezoelectric pickup located under the saddle of each string. When the strings are not being played they are muted, using a magnetic system that prevents "sympathetic vibrations" when playing other strings.

It is based on other similar instruments such as the Chapman Stick or the StarrBoard, whose execution is also performed by pressing the strings on the frets. Some of the most famous performers of this instrument are Stevie Wonder, Jacob Collier, Justin-Lee Schultz or Cory Henry.

In this project, the operation of a G16 Harpejji model is simulated, which has 16 strings and the notes are distributed as in theFigure2.

16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
F#3	G#3	A#3	C4	D4	E4	F#4	G#4	A#4	C5	D5	E5	F#5	G#5	A#5	C6
F3	G3	A3	B3	C#4	D#4	F4	G4	A4	B4	C#5	D#5	F5	G5	A5	B5
E3	F#3	G#3	A#3	C4	D4	E4	F#4	G#4	A#4	C5	D5	E5	F#5	G#5	A#5
D#3	F3	G3	A3	B3	C#4	D#4	F4	G4	A4	B4	C#5	D#5	F5	G5	A5
D3	E3	F#3	G#3	A#3	C4	D4	E4	F#4	G#4	A#4	C5	D5	E5	F#5	G#5
C#3	D#3	F3	G3	A3	B3	C#4	D#4	F4	G4	A4	B4	C#5	D#5	F5	G5
C3	D3	E3	F#3	G#3	A#3	C4	D4	E4	F#4	G#4	A#4	C5	D5	E5	F#5
B2	C#3	D#3	F3	G3	A3	B2	C#4	D#4	F4	G4	A4	B4	C#5	D#5	F5
A#2	C3	D3	E3	F#3	G#3	A#2	C4	D4	E4	F#4	G#4	A#4	C5	D5	E5
A2	B2	C#3	D#3	F3	G3	A2	B2	C#4	D#4	F4	G4	A4	B4	C#5	D#5
G#2	A#2	C3	D3	E3	F#3	G#2	A#2	C4	D4	E4	F#4	G#4	A#4	C5	D5
G2	A2	B2	C#3	D#3	F3	G2	A2	B2	C#4	D#4	F4	G4	A4	B4	C#5
F#2	G#2	A#2	C3	D3	E3	F#2	G#2	A#2	C4	D4	E4	F#4	G#4	A#4	C5
F2	G2	A2	B2	C#3	D#3	F3	G3	A3	B3	C#4	D#4	F4	G4	A4	B4
E2	F#2	G#2	A#2	C3	D3	E3	F#3	G#3	A#3	C4	D4	E4	F#4	G#4	A#4
D#2	F2	G2	A2	B2	C#3	D#3	F3	G3	A3	B3	C#4	D#4	F4	G4	A4
D2	E2	F#2	G#2	A#2	C3	D3	E3	F#3	G#3	A#3	C4	D4	E4	F#4	G#4
C#2	D#2	F2	G2	A2	B2	C#3	D#3	F3	G3	A3	B3	C#4	D#4	F4	G4
C2	D2	E2	F#2	G#2	A#2	C3	D3	E3	F#3	G#3	A#3	C4	D4	E4	F#4

Figure2. Schematic representation of the distribution of the notes in the Harpejji. In white, the positions that correspond to natural notes and, in black, to notes with accidentals.

Given the difficulty in obtaining information directly from the manufacturer, it has been necessary to assume some of the characteristics of the instrument, such as its scale length, which has been estimated at 27” or, what is the same, 68.58 cm. The rest of the parameters of each of the strings have been measured or calculated from a set of Harpejji G16 strings. The results have been collected in theBoard1.

No.	16	fifteen	14	13	12	eleven	10	9	8	7	6	5	4	3	2	1
Gauge [“]	.64	.56	.48	.40	.3. 4	.30	.26	.22	.twenty	.18	.16	.14	.12	.10	.09	.08
Diameter [m]	1,626 E-03	1,422 E-03	1,219 E-03	1,016 E-03	8,636 E-04	7,620 E-04	6,604 E-04	5,588 E-04	5,080 E-04	4,572 E-04	4,064 E-04	3,556 E-04	3,048 E-04	2,540 E-04	2,286 E-04	2,032 E-04
lowest note	C2	D2	E2	F#2	G#2	A#2	C3	D3	E3	F#3	G#3	A#3	C4	D4	E4	F#4
Min frequency [Hz]	65.4	73.4	82.4	92.5	103.8	116.5	130.8	146.8	164.8	185.0	207.7	233.1	261.6	293.7	329.6	370.0
Linear density [kg/m]	1.30 E-02	1.03 E-02	7.64 E-03	5.26 E-03	3.92 E-03	2.86 E-03	2.24 E-03	1.63 E-03	1.03 E-03	1.12 E-03	9.57 E-04	6.73 E-04	5.00 E-04	4.00 E-04	3.00 E-04	2.60 E-04
Tension [N]	104.3	104.5	97.6	84.6	79.5	73.0	72.3	66.2	68.5	72.3	77.6	68.8	64.4	64.9	61.3	67.0
Sound speed [m/s]	89.71	100.70	113.03	126.87	142.41	159.85	179.42	201.39	257.75	253.74	284.82	319.70	358.85	402.79	452.12	507.48

Board1. String parameters of a Harpejji G16.

1.2.Music synthesis methods

Sound synthesis is the technique of generating sound from non-acoustic media, using electronic equipment or software. Depending on the technique used to generate the sound, there are several types of synthesis that are explained below to justify the chosen synthesis method.

1.2.1.additive synthesis

It is the set of techniques in which sound is generated by adding sinusoids of different frequencies. As explained in [3], to generate a specific timbre, a certain number of its harmonics are added to the fundamental frequency. An envelope of different amplitude is applied to each harmonic, so that the decay time of each harmonic is different, just like real instruments, for a more realistic sound.

Another strategy for greater realism is to slowly modulate the frequencies of the sinusoids to slightly vary the pitch of the synthesized note and simulate the imperfections of a real instrument or player. To generate the sound analogically, it is necessary to use a bank of sinusoids with a configurable frequency and amplitude envelope. In theFigure3A schematic block diagram of an additive synthesizer is included, taken from [4].

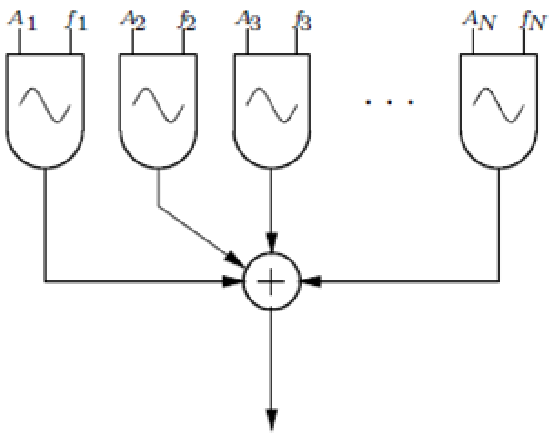


Figure3. Block diagram of an additive synthesizer.

1.2.2.subtractive synthesis

The sound is generated by filtering and amplifying the signal from an oscillator with different waveforms. Subtractive synthesis starts with a signal rich in harmonics and is filtered as appropriate. It is the most common synthesis technique used in analog hardware synthesizers.

The signal generated by one or more oscillators (usually a triangle, sawtooth or square signal) is combined and filtered [5]. The signal from the oscillators is fed into a lowpass, highpass, or bandpass filter, which removes some harmonic content from the original mix. Finally it is introduced into an amplifier that controls the volume of the sound.

Depending on the implementation used, the parameters (filter cutoff frequency, oscillator fundamental frequency, amplifier gain...) can be controlled with envelope generators, which are typically set to start when a note is pressed on the keyboard, or with low frequency oscillators if you want to make these parameters vary slowly.

In theFigure4A possible block diagram of a subtractive synthesizer is observed.

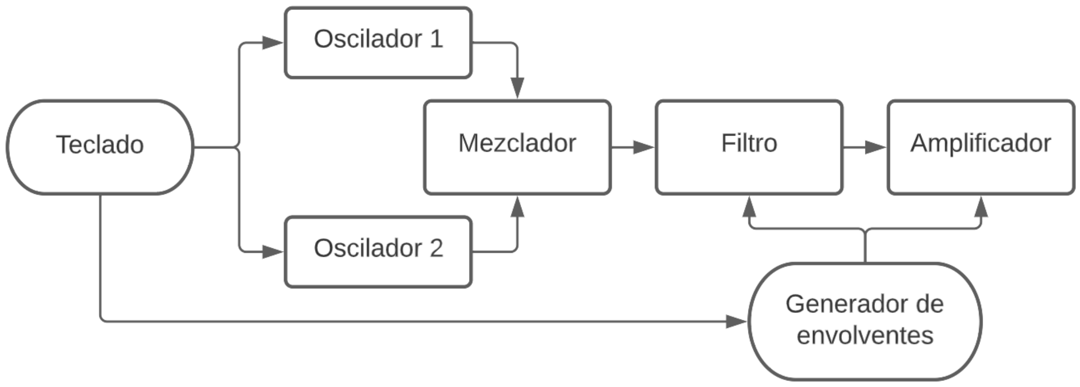


Figure4. Typical block diagram of a subtractive synthesizer.

1.2.3.Frequency Modulation Synthesis

Sound is generated by varying the frequency of a carrier signal according to a second (modulating) signal. Through this modulation, harmonics are generated both above and below the carrier frequency separated by the same interval [6]. By using two to six oscillators, it is possible to generate complex signals that closely resemble the sounds of real instruments.

One of the most famous synthesizers that uses this type of synthesis is the Yamaha DX7, with a characteristic sound of the eighties and that has six oscillators (operators) that combine and modulate each other. In theFigure5the algorithms used by the Yamaha DX7 synthesizer are shown, taken from [7]. In blue the modulating oscillators are indicated, while in green the carriers are represented, which produce the signal that is reproduced at the output of the synthesizer [8]. The drawback of this synthesis method is the lack of correlation between the user-changeable parameters and the sound produced.

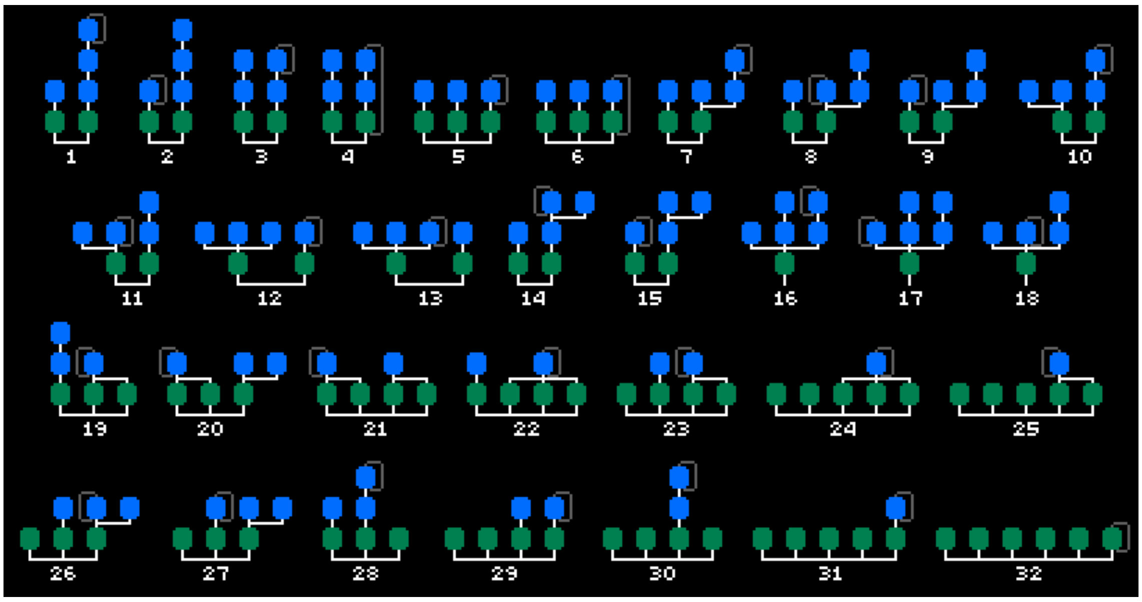


Figure5. Algorithms used in the Yamaha DX7 synthesizer. Each algorithm is composed of the combination of six operators.

1.2.4.Wavetable synthesis

It is the most common digital implementation for oscillators [4]. The waveform of an oscillator period is stored in a "wavetable", instead of being calculated in real time. To reproduce the sound of the oscillator, the information stored in the table is read at a certain speed, which determines the fundamental frequency of the oscillator. It is common to play multiple wavetables at once or to start playing a waveform by pressing a note and gradually transform it into another. Sounds that evolve over time are thus achieved.

Depending on the frequency of the desired note, it may be necessary to interpolate the samples contained in the wavetable to generate a note of a different frequency [9]. In theFigure6, taken from [4], this reading and interpolation process is represented.

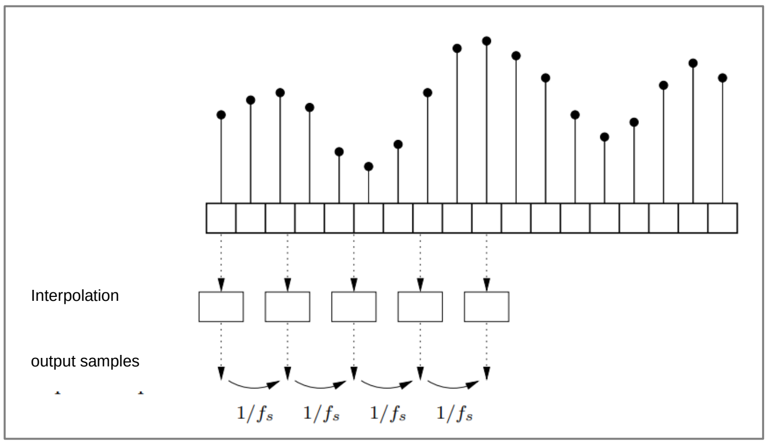


Figure6. "A buffer, which stores a signal, is read at intervals of 1/fs, where fs is the sample rate. Interpolation is used.

1.2.5.granular synthesis

Sound is generated by processes that use “grains” (small windowed segments) of the audio signal. The audio is divided into small chunks (from 1 to 100 ms), which are played by altering their order, playback speed, or by playing several granules at the same time [10].

In [11] two types of granular synthesis are distinguished. The first type, synchronous, is based on the repetition of the granules at a regular frequency, in order to reproduce sounds with a certain pitch. The second type, asynchronous, plays the granules randomly to produce “sound textures”.

1.2.6.Synthesis by physical modeling

The synthesis by physical modeling can be defined, according to [12], as the sound generation method in which a mathematical model is used to simulate the physical source of the sound.

In general, a set of differential equations is used that describe air pressure, the displacement of strings or membranes, etc. To calculate the sound produced by the instrument, the set of equations is solved by approximating them to a numerical solution to obtain an output signal given an input excitation.

To avoid solving the differential equation in real time, it is possible to use techniques such as the waveguide that reflect the behavior of the differential equation through the use of buffers, filters and amplifiers.

1.3.Synthesis methods by physical modeling

1.3.1.Waveguide

One of the most common implementations of synthesis by physical modeling is through the use of waveguides, which are made up of delay lines and filters, to simulate the transmission and reflection of sound in air, on a string, a bar, etc. etc In theFigure7the configuration proposed in [13] for the synthesis of the vibration in an ideal string is shown. This structure can be obtained from the D'Alembert solution to the wave equation, which is explained in more detail in Chapter2and which supposes a traveling wave towards each end of the string.

The initial conditions for the string are entered into the two delay lines and shifted toward each end. A phase inversion occurs at each end of the string (gain -1). The result is obtained by adding the output of the two delay lines at a point on the chord.

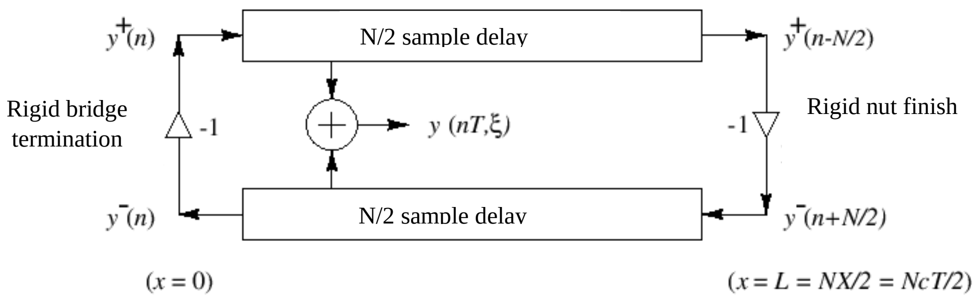


Figure7. Implementation of the ideal wave equation on a string by means of a waveguide.

To simulate air absorption (constant at all frequencies), the gain at the end of the string is reduced below unity, maintaining the phase shift.

In addition, a filter can be introduced at one end of the string (low pass in this case) which causes the losses to be greater at high frequencies, so that the low frequencies remain vibrating on the string for longer.

1.3.2.Differential Equation Approximation

Although it is more computationally expensive than using waveguides, the differential equation can be directly implemented, discretizing it using finite differences. If all the terms of the differential equation are added, it gives rise to a damped vibration whose losses depend on the frequency. Once the differential equation has been implemented, it is only necessary to apply some boundary conditions and some initial conditions to the string and take the vibration at a point on the string to generate the sound.

The implementation process using finite differences of the differential equation is explained in detail in the section3.4.

1.4.Advantages and disadvantages of synthesis by physical modeling

The main advantage of this method is the sound quality achieved compared to other synthesis methods. Once the basic elements of an instrument are known (strings, membranes, plates...) it is easy to extrapolate it to another without making too many modifications.

Through physical modeling synthesis it is possible to emulate both acoustic instruments and electric and even electronic instruments by applying, for example, real-time models of filters, amplifiers and oscillators to model classic analog synthesizers. Another advantage of this method is that it allows to generate instruments with impossible physical characteristics: extraordinary dimensions and proportions, unusual materials, etc.

There are currently numerous examples of virtual instruments on the market that use this type of synthesis to reproduce sounds of real instruments (such as the bass emulation plugin MODO Bass from IK Multimedia, or SWAM Strings, from Audio Modeling that emulates string instruments rubbed), as well as some examples of experimental synthesizers that do not seek to simulate the sound of any specific instrument, but allow the combination of mechanical, acoustic and electrical elements to generate new sounds (a good example of this type of synthesizer can be the synthesizer of Applied Acoustics Systems Chromaphone 3 Percussion).

In many cases, as in the examples mentioned in the previous paragraph, sounds indistinguishable from those of the real emulated sound are achieved, also allowing on occasions to alter the properties of the instrument, being able to produce sounds that are not possible in real instruments due to physical limitations. of the materials or medium of sound transmission.

One of the problems of this type of synthesis is the high computational cost involved. As Marc-Pierre from Applied Acoustics Systems explains in [14], when the program is run there are no pre-recorded samples, rather the sound must be generated while it is running, which requires some CPU usage. However, if it is well optimized, it is possible to achieve very good results with an acceptable computational cost.

1.5.design specifications

The objective of the project is the development of a software synthesizer in the form of a plugin that emulates the sound of Harpejji through physical modeling of the instrument's strings.

The plugin is developed in VST3 (Virtual Studio Technology version 3) so that it can be run on any standard digital audio workstation, both on Windows and MacOS.

In the preliminary phase the following design specifications were established for the synthesizer:

- The virtual instrument would take the form of a cross-platform plugin that could be run on any standard VST-compatible digital audio workstation, both on Windows and MacOS.

- The plugin would emulate the sound of a Harpejji through synthesis by physical modeling of the strings of the instrument.
- The plugin would take MIDI instructions from the host program as its input and would provide the emulated sound of the instrument in mono format in its audio output.
- The plugin would work in real time, with no appreciable latency for the user.
- The virtual instrument would be sensitive to the velocity of MIDI notes input to it.
- The user could modify the physical parameters of the instrument through the graphical interface of the plugin.
- The instrument would be polyphonic with a number of voices to be determined based on performance.

2. string vibrations

In order to model the strings of the Harpejji it is first necessary to know the properties of the strings in general and how vibrations propagate in them. This chapter explains how vibration is produced in the strings, following the theoretical development of [15] in the investigation.

Only transverse movements are considered, perpendicular to the direction of the chord. Although there are also longitudinal and torsional movements in the strings, their importance in the sound can be considered negligible compared to the contribution of the transversal ones.

2.1.the perfect rope

It begins by considering a string as a set of masses and springs, as shown schematically in theFigure8. A number of masses is added such that the distance between them is reduced to and the total number of masses is $.dxn = L/dx$

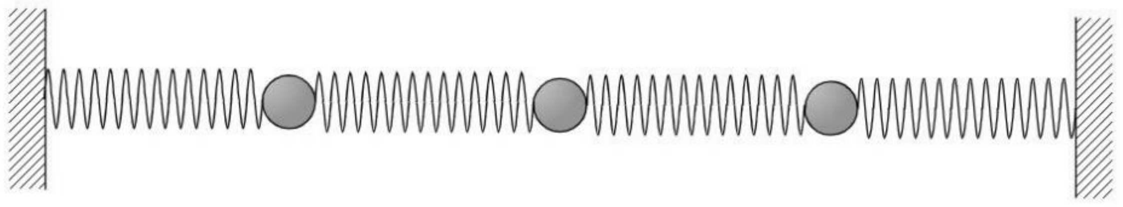


Figure8. Representation of a string as a system of masses and springs.

For every mass added to the system, a new transverse eigenmode appears in the string, as represented in theFigure9.

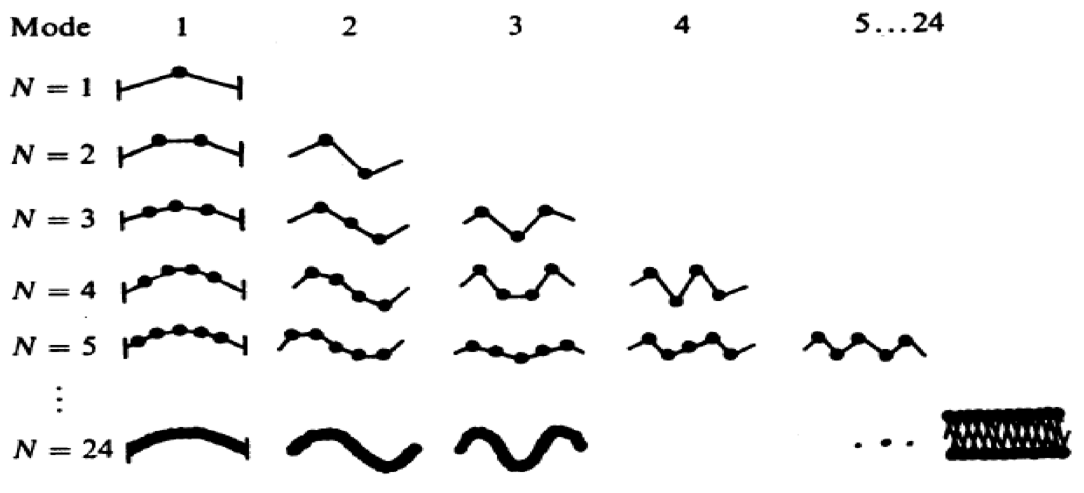


Figure9. Vibration modes of the string as masses are introduced into the system.

From the expression for the tension in the string and applying the Taylor series and Newton's second law (the step-by-step development can be found in [15]) we arrive at the equation(1), which represents the movement of transverse waves in a vibrating wave.

$$\frac{\partial^2 y}{\partial t^2} = \frac{T}{\mu}\,\frac{\partial^2 y}{\partial x^2} \tag{1}$$

where is the direction parallel to the longitudinal axis of the string, is the transverse direction, is the tension in Newtons, and is the linear density of the string, in kg/m.$xyT\mu$

The general solution for this wave equation, proposed by D'Alembert, is that of two waves traveling at the same speed in opposite directions of the string.$f_1 f_2$

$$y = f_1(ct - x) + f_2(ct + x) \tag{2}$$

where is the speed of propagation of waves on the string.$c = \sqrt{T/\mu}$

2.2.Boundary conditions.

In the case of Harpejji strings, unlike other instruments with movable bridges (fiddle or banjo, for example), boundary conditions can be approximated to those of a string with fixed ends, in the same way as of an electric guitar. This means that the displacement in a string of length is zero at both the point and the and, therefore, the impedance at the two ends, , approaches infinity.$Lx = 0x = LZ$

Using D'Alembert's solution and applying the fixed end condition for the pointis obtained:$x = 0$

$$y = 0 = f_1(ct - 0) + f_2(ct + 0) \tag{3}$$

So, at the end of the string:

$$f_1(ct) = -f_2(ct) \tag{4}$$

This is easily verifiable by calculating the reflection coefficient, through the equation$R$(5).

$$R = \frac{Z_0 - Z}{Z_0 + Z} \tag{5}$$

where , is the characteristic impedance of the string, is the impedance of the end of the string, and is the ratio between the amplitude of the incident wave and the reflected wave.$Z_0 Z R = A_r/A_i$

For infinite end impedance a reflection coefficient is obtained$R = -1$.

$$R = \lim_{Z \to \infty} \frac{Z_0 - Z}{Z_0 + Z} = \frac{-\infty}{\infty} = -1 \tag{6}$$

This means that a phase reversal will occur as the wave reflects off the fixed end. This process of reflection is represented in theFigure10.

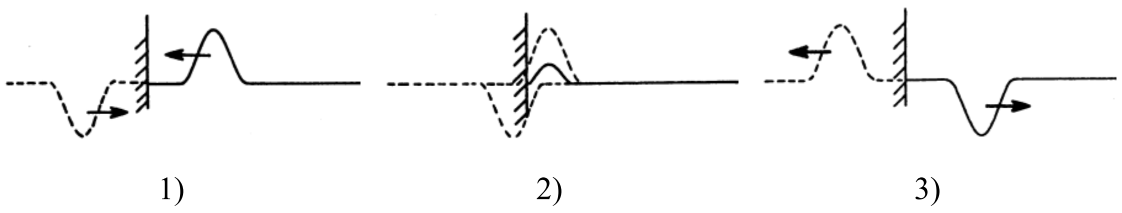


Figure10. Reflection of a wave at one of the fixed ends of a string.

2.3.Harmonic Solutions of the Wave Equation

To study the propagation of harmonic motions through the string, we consider the functions y that move towards each side of the string, formed by a sine term and a cosine term.$f_1 f_2$

$$y(x,t) = A\,\mathrm{sen}(\omega t - kx) + B\cos(\omega t - kx) + C\,\mathrm{sen}(\omega t + kx) + D\cos(\omega t + kx) \tag{7}$$

where is the wave number.$k = \omega/c$

If we apply the boundary conditions for a chord of length and ends fixed at y, we obtain the equation$L(x = 0)(x = L)$(8).

For fixed end at:$(x = 0)$

$$y(0,t) = 0 = A\,\mathrm{sen}(\omega t) + B\cos(\omega t) + C\,\mathrm{sen}(\omega t) + D\cos(\omega t) \tag{8}$$

It must necessarily be true that and , so that:$A = -CB = -D$

$$y = A[\mathrm{sen}(\omega t - kx) - \mathrm{sen}(\omega t + kx)] + B[\cos(\omega t - kx) - \cos(\omega t + kx)] \tag{9}$$

Using the cosine and sine formulas, the sum and difference can be reduced to the equation(10).

$$y = 2[A \cos \omega t - B \operatorname{sen} \omega t] \operatorname{sen} kx \tag{10}$$

For the extreme, the second condition must be fulfilled. For this it is necessary that , or what is the same, . This condition restricts the values of such that , or in terms of frequency, .$(x = L) y(L,t) = 0 \operatorname{sen}(kL) = 0\, \omega L/c = n\pi\omega\omega_n = n\pi c/L f_n = n(c/2L)$

These frequencies represent the proper modes of vibration of the string, which respond to the expression:

$$y_n(x,t) = (A_n \operatorname{sen} \omega_n t + B_n \cos \omega_n t) \operatorname{sen} k_n x \tag{11}$$

These modes are harmonic because each frequency is times .$f_n n f_1 = c/2L$

The general solution of a vibrating string with fixed ends can be written as the sum of its eigenmodes:

$$y(x,t) = \sum_n (A_n \operatorname{sen} \omega_n t + B_n \cos \omega_n t) \operatorname{sen} k_n x \tag{12}$$

2.4.Losses

Until now, the rope has been considered as an ideal system, without energy losses. In a real string, however, energy losses occur due to different factors, which translate into vibration attenuation. The three main factors that cause losses in strings are air, string material, and energy transmission to other vibrating systems at the string ends.

To reproduce viscous damping, such as that of air, it is common to include an attenuation term that opposes motion in the differential equation and depends on a constant , which determines how fast the waves on the string are attenuated.σ_0

$$\frac{\partial^2 y}{\partial t^2} = \frac{T}{\mu} \frac{\partial^2 y}{\partial x^2} - \sigma_0 \frac{\partial y}{\partial t} \tag{13}$$

The solution to this new equation is also harmonic but multiplied by a negative exponential factor dependent on .σ_0

$$y(x,t) = e^{-\sigma_0 t} \sum_{n=1}^{\infty} (A_n \operatorname{sen} \omega_n t + B_n \cos \omega_n t) \operatorname{sen} k_n x \tag{14}$$

This new term causes a constant attenuation at all frequencies. On a real string, however, the higher harmonics are attenuated before the lower ones. To reproduce this behavior, a new frequency-dependent attenuation term is introduced into the wave equation. The change in curvature of the string is chosen so that at higher frequencies the change in curvature is greater and more attenuation occurs.

The complete equation with the two attenuation terms is included in(15).

$$\frac{\partial^2 y}{\partial t^2} = \frac{T}{\mu}\frac{\partial^2 y}{\partial x^2} - \sigma_0 \frac{\partial y}{\partial t} + \sigma_1 \frac{\partial}{\partial t}\frac{\partial^2 y}{\partial x^2} \tag{15}$$

The decay time varies with frequency as seen in theFigure11, taken from [4] and is controlled with the parameters and .$\sigma_0 \sigma_1$

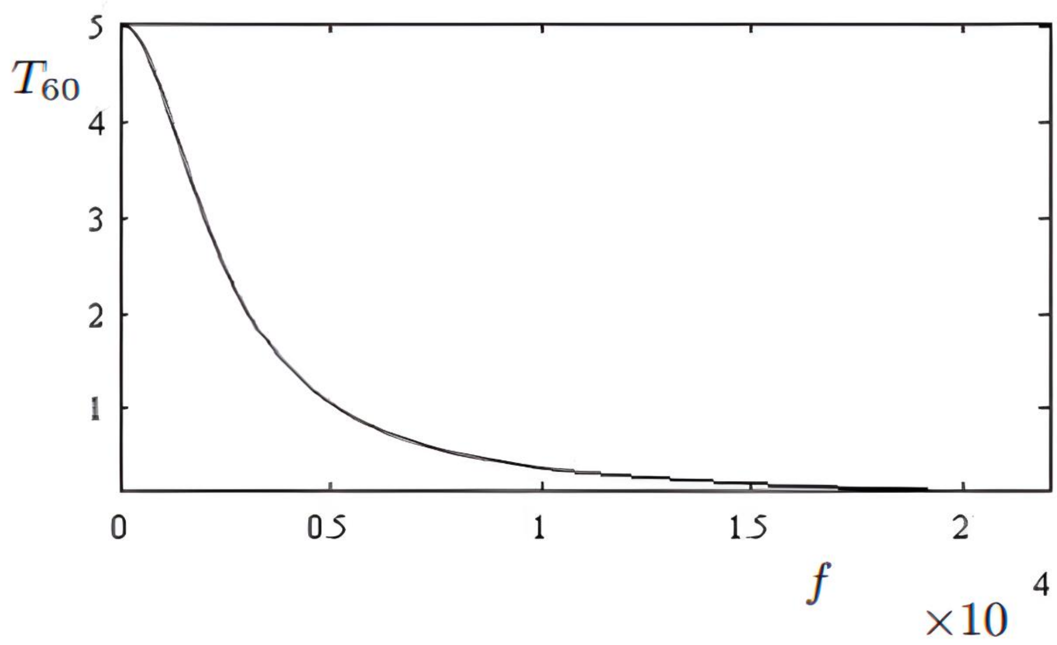


Figure11. Plot of T60 versus frequency, with T60 = 3s at 2500 Hz and T60 = 1s at 5 kHz.

2.5.Rigidity

To some extent the strings of musical instruments behave like rigid bars. When a bar is bent, the outer part lengthens while the inner part compresses. From the force on the rope[fifteen], represented in the equation(16)the differential equation term for the stiffness can be obtained(17).

$$F = \frac{\partial M}{\partial x} = -ESK^2 \frac{\partial^3 y}{\partial x^3} \tag{16}$$

$$\frac{\partial^2 y}{\partial t^2} = -\frac{EK^2}{\rho}\frac{\partial^4 y}{\partial x^4} \tag{17}$$

Where:

- $K = a/2$is the radius of gyration for cylindrical shapes and is the radius of the chord or bar.a
- $E$is the Young's Modulus or longitudinal modulus of elasticity, which characterizes the elastic behavior of a material [16].
- $S$is the cross-sectional area of the string.
- ρ, the volumetric density of the rope or bar.

Add this term to the differential equation and thus obtain the complete differential equation for the vibration of a string.

$$\frac{\partial^2 y}{\partial t^2} = \frac{T}{\mu}\frac{\partial^2 y}{\partial x^2} - \sigma_0\frac{\partial y}{\partial t} + \sigma_1\frac{\partial}{\partial t}\frac{\partial^2 y}{\partial x^2} - \frac{EK^2}{\rho}\frac{\partial^4 y}{\partial x^4} \tag{18}$$

This term introduces a difference in the speed of propagation of the waves on the string or bar for the different frequencies, that is, it introduces dispersion.

3. plugin development

This chapter explains the plugin development process step by step: the choice of technology used for development and the final format of the plugin and the general operation of the program are explained. The process of approximation of the wave equation by means of finite differences and its transfer to C++ code is detailed. The different types of excitation that have been considered for the string and their effect on the sound of the instrument are listed below. Finally, all the elements that make up the plugin are detailed, both the backend and the user interface, and its operation is explained.

3.1.Virtual Studio Technology (VST)

We have chosen to develop the plugin in VST (Virtual Studio Technology) format due to its wide compatibility with most digital audio workstations (DAWs).

VST [17] is an interface that allows communication between audio synthesizers and effects plugins and audio editing, sequencing and recording software. It was developed by Steinberg and released in 1996, completely transforming the audio recording and editing processes. At present, plugins have almost completely replaced hardware equipment in recording studios due to their practicality and lower cost.

Most VST plugins can be classified as VST instruments (VSTi) or audio effects (VSTfx), although there are also plugins for MIDI effects, meters, spectrum analyzers, etc. VST plugins are controlled by a graphical user interface, which is presented in windows.

The main advantage of VST technology is that it allows software from different companies and functionalities to be connected to a single recording system within the personal computer. This makes it a very versatile and flexible technology.

3.2.JUCE bookstore

It has been decided to use the JUCE library to develop the plugin using C++ as programming language. This open source library for non-commercial use takes care of most of the input, output and sampling processes, simplifying many processes and limiting the operations that are necessary to perform on a sample-by-sample basis.

JUCE is configured to generate a synthesizer application template, using the JUCE Synthesiser class [18], which generates the basic functions to receive MIDI instructions at its input and output whatever is placed in the plugin's output buffer. The program is divided into four scripts and in each of them the instructions related to a part of the program are written: PluginProcessor, PluginEditor, SynthesizerVoice and SynthesizerSound.

- PluginProcessor: in this script the backend of the plugin is programmed. The audio processes are included in it, the user-controllable parameters are created and, in the case of a synthesizer, the SynthesizerVoice type objects (each of the synthesizer voices) are generated. The audio processing that affects all the voices is also carried out in it; in this case, the gain and filtering of the set of all voices is performed.

- PluginEditor: This contains instructions for the user interface and graphical aspects of the plugin. In the case of the synthesizer, the potentiometers that control the parameters previously created in PluginProcessor are created in this script and their position within the plugin window is established. In this case, the instructions necessary to visualize the vibration of the strings and represent them on a virtual Harpejji are also included in this document.

- SynthesiserVoice: the bulk of the operations related to the physical modeling of the Harpejji string are performed here. Upon receiving a MIDI note, the plugin creates a string with the proper characteristics (length, tension, and linear density) to match the frequency of the pressed note, and sets initial conditions on it, depending on the velocity of the keyboard. . The chord position is calculated for subsequent samples until the level falls below a threshold. Then that voice fades away.

- SynthesizerSound: in this script only some parameters necessary for the operation of the program are established.

3.3.General operation of the plugin

When the plugin is opened in a digital audio station, it is initialized, creating the parameters that the user will have access to from the graphical interface, as well as the low-pass filter that will serve as tone control and that is explained in detail in point3.10. Next, an object of type SynthesiserSound is created and each of the synthesizer voices, of type SynthesiserVoice, are added, leaving the plugin with the structure of theFigure12.

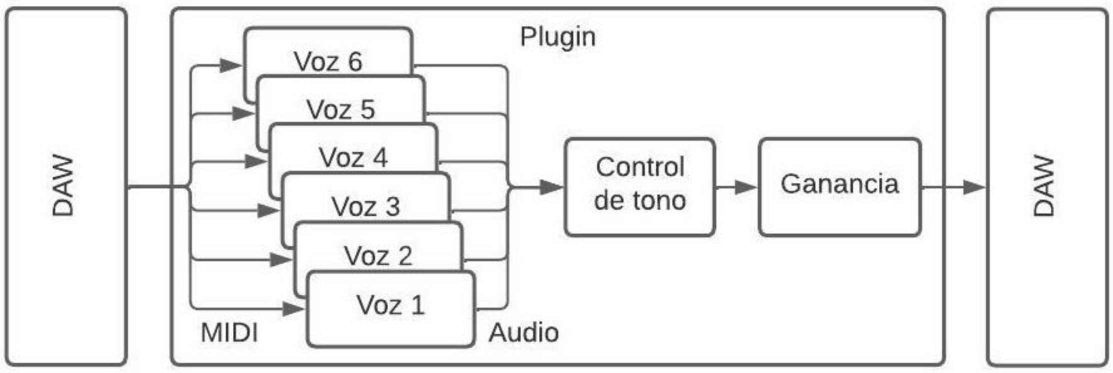


Figure12. Block diagram of the plugin.

Before starting to play audio, it is necessary to establish some general parameters such as the sampling frequency, the frame size and the number of output channels with which you are going to work. The host program provides this information to the plugin and it is stored to correctly generate audio signal that the DAW can play.

Once the process that is in charge of audio processing is initialized, the graphical user interface is initialized, which is a secondary process with lower priority than the main audio process. Its refresh rate is set and the plugin's background image is displayed, on which the running strings and rotary controls are drawn. All these processes related to the graphical user interface are explained in more detail in the section3.12.

With the audio processing and user interface initialized the plugin is waiting to receive MIDI messages. Upon receiving a MIDI note, JUCE takes care of looking for a free voice (that is not currently playing a note) and calls the startNote() method of that voice. Within this method it is checked that the note is in the range of the Harpejji G16 (C2 – C6) and, if so, the model of the string is generated and the initial conditions are established on it. Otherwise, the voice is released.

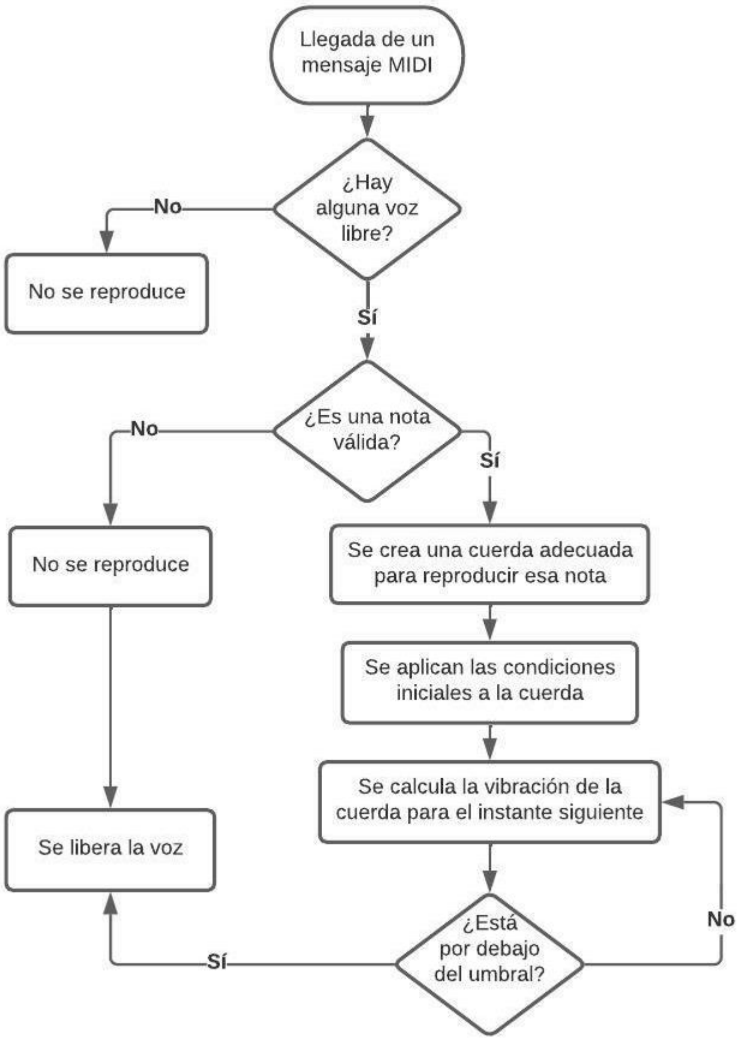

Llegada de un mensaje MIDI
¿Hay alguna voz libre?
No
No se reproduce
Sí
¿Es una nota válida?
No
Sí
No se reproduce
Se crea una cuerda adecuada para reproducir esa nota
Se aplican las condiciones iniciales a la cuerda
Se calcula la vibración de la cuerda para el instante siguiente
Se libera la voz
¿Está por debajo del umbral?
Sí
No

Figure13. Flowchart of each synthesizer voice.

To generate the string model, the spatial sampling step is calculated from the frequency of the received note. Depending on the position (string/fret combination) in which the note is to be "played", a string of a certain tension and length is generated. Once the length in sampling points that the string must have has been calculated, three vectors of this size are created that will contain the position of the string at the current instant (y), at the previous instant of execution (yPrev) and at the following instant. (yNext).

The initial conditions on the string are then applied. To this end, one of the excitation functions that are explained in the section3.5adapted to the calculated chord length. This function will assume the initial velocity of the string when the note is pressed, which is corrected in amplitude so that it depends on the MIDI velocity of the received note (detailed in point3.6).

Knowing the initial velocity in the string and knowing that the displacement at the moment of its pulsation is null throughout the string, the position of the string at the next instant of execution is calculated sample by sample. For this, the differential equation of the approximated wave by means of finite differences is used. In it, the position of the entire string is entered at the current instant and at the previous instant of execution, and from these the position at the next instant is calculated. The process of approximation of the wave equation by means of finite differences is explained in depth in the section3.4.

The position of the string at the current time, at the reading position of the string, is then taken and entered into the plugin's output buffer. The process is repeated sample by sample until the frame corresponding to each of the active voices is completed and the contributions of all voices are added to the output of the plugin.

It is important to note that the MIDI note does not have to arrive at an instant that coincides with the start of a frame. Upon arrival of a MIDI note, JUCE provides the sample number in which it occurs. Only samples between this and the buffer size for the first frame are processed. The following frames are processed completely.

All the active voices continue to be processed until the level of the RMS level detector of any of them (developed in the section3.8) falls below the threshold. At that moment the note is stopped playing and the voice is released.

Before applying the plugin's output buffer, it is processed, filtering it and applying the corresponding gain to the tone and gain controls (sections3.10and3.11).

The graphical interface is drawn 60 times per second by calling the paint() method; this includes the background (.JPG image), the strings of the instrument, and the rotary controls that the user can interact with. These rotary controls are linked to some parameters (tension, sustain, tone and gain) that are passed from the interface to the main audio process which, in turn, passes them to each of the voices so that the necessary changes are made in the characteristics of the strings.

If any of the voices is active, the vibration of the string is drawn on the graphical interface at the position of the corresponding string. To do this, each of the voices provides the vibration of the string to the main audio process approximately once every 5 ms and the main audio process passes it to the graphical interface to draw it when the paint() method is called (every 16 ms).

3.4.Finite difference approximation of the wave equation

The program code calculates the position of the string for the next time instant from the position of the string at the current time and the previous time. For this, it is necessary to adapt the wave equation, discretizing it and later translating it into C++ code. The following finite difference approximations are used to discretize the differential equation taken from [19].

First derivative. Backwards difference:

$$\frac{\partial f}{\partial g} \approx \frac{f_{i-1} - f_i}{\Delta g} \tag{19}$$

First derivative. Forward difference:

$$\frac{\partial f}{\partial g} \approx \frac{f_i - f_{i+1}}{\Delta g} \tag{20}$$

Second derivative. Core Difference:

$$\frac{\partial^2 f}{\partial g^2} \approx \frac{f_{i+1} - 2f_i + f_{i-1}}{(\Delta g)^2} \tag{21}$$

Using these expressions, each of the terms of the differential equation is individually discretized to later combine them into a single expression.

$$\frac{\partial^2 y}{\partial t^2} = \frac{T}{\mu}\frac{\partial^2 y}{\partial x^2} - 2\sigma_0 \frac{\partial y}{\partial t} + \sigma_1 \frac{\partial}{\partial t}\frac{\partial^2 y}{\partial x^2} \tag{22}$$

The objective is to calculate the position of the string at the next sampling instant, which is denoted in the equations as . To this end, all the terms of the equation are calculated individually and once the expressions of all of them are obtained, the expression of is found, combining them.$y(x, t + 1)y(x, t + 1)$

3.4.1.Second derivative term

To discretize the first term, to the left of the equation, the expression for the second derivative using central differences is directly applied$\left(\frac{\partial^2 y}{\partial t^2}\right)$(21), which gives rise to the equation(23).

$$\frac{\partial^2 y}{\partial t^2} \approx \frac{y(x, t+1) - 2y(x, t) + y(x, t-1)}{(\Delta t)^2} \tag{23}$$

3.4.2.Ideal chord term

The second term , which represents the wave equation of an ideal string, is first approximated by applying the same expression of the equation$\left(\frac{T}{\mu}\frac{\partial^2 y}{\partial x^2}\right)$(21) to the second derivative of with respect to . Thus, the equation is obtainedyx(24).

$$\frac{\partial^2 y}{\partial x^2} \approx \frac{y(x+1,t) - 2y(x,t) + y(x-1,t)}{(\Delta x)^2} \tag{24}$$

That when adding the factor it remains as in the equation$\frac{T}{\mu}$(25).

$$\frac{T}{\mu}\frac{\partial^2 y}{\partial x^2} \cong \frac{T}{\mu}\,\frac{y(x+1,t) - 2y(x,t) + y(x-1,t)}{(\Delta x)^2} \tag{25}$$

3.4.3.Linear dimming term

The third term , which corresponds to the linear attenuation, equal to all frequencies, can be approximated using either of the two expressions$\left(-2\sigma_0\frac{\partial y}{\partial t}\right)$(19) and(20) for the first derivative. When applied to the variables of the wave equation, they respectively result in the equations(26) and(27).

$$\frac{\partial y}{\partial t} \approx \frac{y(x,t-1) - y(x,t)}{\Delta t} \tag{26}$$

$$\frac{\partial y}{\partial t} \approx \frac{y(x,t) - y(x,t+1)}{\Delta t} \tag{27}$$

Since it is not known (it is the sample at the next execution instant and the target of all these calculations), the second expression$y(x,t+1)$(27) is not useful in this case. The backward finite difference approximation is used, which employs the previous sample. With this and adding the factor , the expression for the linear attenuation term becomes:$-2\sigma_0$

$$-2\sigma_0\frac{\partial y}{\partial t} \cong -2\sigma_0\frac{y(x,t-1) - y(x,t)}{\Delta t} \tag{28}$$

3.4.4.Frequency dependent damping term.

To approximate the third term , which corresponds to the frequency-dependent attenuation; First, the expression of the second derivative using central differences is used$\left(\sigma_1 \frac{\partial}{\partial t} \frac{\partial^2 y}{\partial x^2}\right)$(21), from which we obtain the equation(29). Next, the expression of the first derivative and finite differences backwards is applied to this(19), obtaining the equation(30).

$$\frac{\partial^2 y}{\partial x^2} \approx \frac{y(x+1,t) - 2y(x,t) + y(x-1,t)}{(\Delta x)^2} \tag{29}$$

The following pages have been turned to landscape to improve the readability of the equations.

$$\frac{\partial}{\partial t}\frac{\partial^2 y}{\partial x^2} \approx \left[\frac{y(x+1,t)-2y(x,t)+y(x-1,t)}{(\Delta x)^2} - \frac{y(x+1,t-1)-2y(x,t-1)+y(x-1,t-1)}{(\Delta x)^2}\right]/\Delta t \tag{30}$$

Finally, the attenuation factor is added, , and the complete expression for this term is left as:σ_1

$$\sigma_1 \frac{\partial}{\partial t}\frac{\partial^2 y}{\partial x^2} \cong \sigma_1 \left[\frac{y(x+1,t)-2y(x,t)+y(x-1,t)-y(x+1,t-1)+2y(x,t-1)-y(x-1,t-1)}{(\Delta x)^2\Delta t}\right] \tag{31}$$

3.4.5.full wave equation

To obtain the position of the string at the next instant of execution, the expressions obtained for each of the terms of the wave equation are combined(22).

$$\frac{\partial^2 y}{\partial t^2} = \frac{T}{\mu}\frac{\partial^2 y}{\partial x^2} - 2\sigma_0\frac{\partial y}{\partial t} + \sigma_1\frac{\partial}{\partial t}\frac{\partial^2 y}{\partial x^2} \tag{22}$$

The expressions of the equations are combined(25),(28) and(31) to obtain the equation32).

$$\begin{aligned}\frac{y(x,t+1)-2y(x,t)+y(x,t-1)}{(\Delta t)^2} &\cong \frac{T}{\mu}\,\frac{y(x+1,t)-2y(x,t)+y(x-1,t)}{(\Delta x)^2} - 2\sigma_0\frac{y(x,t-1)-y(x,t)}{\Delta t} \\ &+ \sigma_1\left[\frac{y(x+1,t)-2y(x,t)+y(x-1,t)-y(x+1,t-1)+2y(x,t-1)-y(x-1,t-1)}{(\Delta x)^2\Delta t}\right]\end{aligned} \tag{32}$$

Finally, it can be solved to obtain the expression for the next string position from the previous samples:$y(x, t+1)$

$$\begin{aligned} y(x,t+1) = {} & \frac{(\Delta t)^2}{(\Delta x)^2}\frac{T}{\mu}\left(y(x+1,t) - 2y(x,t) + y(x-1,t)\right) + 2y(x,t) - y(x,t-1) \\ & - 2\sigma_0\,\Delta t[y(x,t-1) - y(x,t)] \\ & + 2\sigma_1\frac{\Delta t}{(\Delta x)^2}[y(x+1,t) - 2y(x,t) + y(x-1,t) - y(x+1,t-1) + 2y(x,t-1) \\ & - y(x-1,t-1)] \end{aligned} \tag{33}$$

This expression can be translated directly into C++ code as shown in theCode1. For this, three size vectors are used`x`, equal to the length in samples of the chord, which contain the displacement at each point of the chord at the current instant (vector`and)`, at the previous execution instant (`yPrev)` and in the next (`yNext).dt`is the temporal sampling period, equal to`1/fs`and`dx`, the spatial sampling period, or the distance between one position on the string and the next.

The displacement of the rope in all its positions is calculated, from`(x= 1) to (x=x-1)`since the ends are fixed and their displacement is always zero. Once the displacement in the entire string has been calculated, the sample is read in the reading position,`xRead`, close to one of the fixed ends because that's where the piezo pickups are located on the actual instrument, and feed into the output buffer of that voice. Once the sample has been read, the three vectors are updated: the current position,`and`, happens to be`yPrev`and the position at the next execution instant becomes the current position.

This process is repeated for each sample until the synth voice buffer is full.

The code related to the calculation of the differential equation is included in theCode1.

```
for (int s = 0; s < synthBuffer.getNumSamples(); s++) {
// Cálculo de la posición de la cuerda en el sample siguiente
      for (int x = 1; x < X-1; x++) {
            //Ecuación de onda de la cuerda
            yNext[x] = (T/mu) * (y[x - 1] - 2.0f * y[x] + y[x + 1]) * (powf(dt,2) / powf(dx,2)) + 2 * y[x] - Prev[x]
            // Atenuación lineal
                  - 2.0f * s0 * (y[x] - yPrev[x]) * dt
            // Atenuación dependiente de la frecuencia
                  + 2.0f * s1 * (y[x + 1] - 2.0f * y[x] + y[x - 1] - yPrev[x + 1] + 2.0f * yPrev[x] - yPrev[x - 1])
                  * dt / powf(dx,2);
      }
      // Se añade al buffer el desplazamiento de la cuerda en el punto de lectura.
      synthBuffer.addSample(0, s, y[xRead]);
      yPrev = y;
      y = yNext;
}
```

Code1. Implementation of the wave equation through finite differences.

3.4.6.stiffness term

Finally, the term dependent on the stiffness of the chord can be added to these terms.(34). As explained in the section2.5this term causes a dependence on the speed of propagation of the waves on the string according to their frequency, that is, it introduces dispersion. This term is not very relevant to strings with normal tension and mass, but it has been included to add realism to extreme cases outside of the actual physical characteristics of the materials.

$$\frac{\partial^2 y}{\partial t^2} = -\frac{EK^2}{\rho}\frac{\partial^4 y}{\partial x^4} \tag{34}$$

To approximate it by finite differences, the following expressions are used for the fourth derivative:

Central:

$$\frac{\partial^4 y}{\partial x^4} \cong \frac{y(x+2,t) - 4y(x+1,t) + 6y(x,t) - 4y(x-1,t) + y(x-2,t)}{(\Delta x)^4} \tag{35}$$

Forward:

$$\frac{\partial^4 y}{\partial x^4} \cong \frac{y(x+4,t) - 4y(x+3,t) + 6y(x+2,t) - 4y(x+1,t) + y(x,t)}{(\Delta x)^4} \tag{36}$$

Backward:

$$\frac{\partial^4 y}{\partial x^4} \cong \frac{y(x-4,t) - 4y(x-3,t) + 6y(x-2,t) - 4y(x-1,t) + y(x,t)}{(\Delta x)^4} \tag{37}$$

The central finite difference expression is used for all positions on the chord except those at the extremes, where there is no next or previous position. In these cases, backward or forward differences are used depending on which end of the rope we are at.$(x = x + 1)$ $(x = x - 1)$

The full expression for the stiffness term remains as in Eqs.(38),(39)and(40)using central difference, forwards and backwards respectively.

Central:

$$-\frac{EK^2}{\rho}\frac{\partial^4 y}{\partial x^4} \cong -\frac{EK^2}{\rho}\frac{y(x+2,t) - 4y(x+1,t) + 6y(x,t) - 4y(x-1,t) + y(x-2,t)}{(\Delta x)^4} \tag{38}$$

Forward:

$$-\frac{EK^2}{\rho}\frac{\partial^4 y}{\partial x^4} \cong -\frac{EK^2}{\rho}\frac{y(x+4,t) - 4y(x+3,t) + 6y(x+2,t) - 4y(x+1,t) + y(x,t)}{(\Delta x)^4} \tag{39}$$

Backward:

$$-\frac{EK^2}{\rho}\frac{\partial^4 y}{\partial x^4} \cong -\frac{EK^2}{\rho}\frac{y(x-4,t) - 4y(x-3,t) + 6y(x-2,t) - 4y(x-1,t) + y(x,t)}{(\Delta x)^4} \tag{40}$$

In the final version of the plugin, the calculation of this stiffness term has been removed due to its high computational cost and its negligible impact on the sound.

3.5.string excitation

Once the behavior of the string has been programmed, it is enough to apply some initial conditions to it each time a MIDI message is received. To do this, once it has been decided on which string the note is going to be played, the speed of sound on it is calculated (depending on the linear density and the tension of the string and, therefore, different for each string). and a string with the appropriate length is “generated” to reproduce the frequency of the note with that speed of transmission on the string, using the expression(42), which in turn is obtained from(41).

$$c_0 = L_{esc} \cdot 2f_{min} \tag{41}$$

$$L_{cuerda} = \frac{c_0}{2 \cdot f_{nota}} \tag{42}$$

After obtaining the length of the chord, it is divided by the spatial sampling step () to obtain the number of points that are considered to be of the chorddx(43). Next, initial conditions are established on the string by applying an initial displacement and velocity to each point on the string.

$$X = \frac{L_{cuerda}}{dx} \tag{43}$$

In the case of plucked string instruments (such as the guitar, mandolin, or harp, for example), the initial conditions applied to the string can be approximated by a triangular initial displacement, as in theFigure14 and an initial velocity of zero caused by acting on the string the finger or the pick.

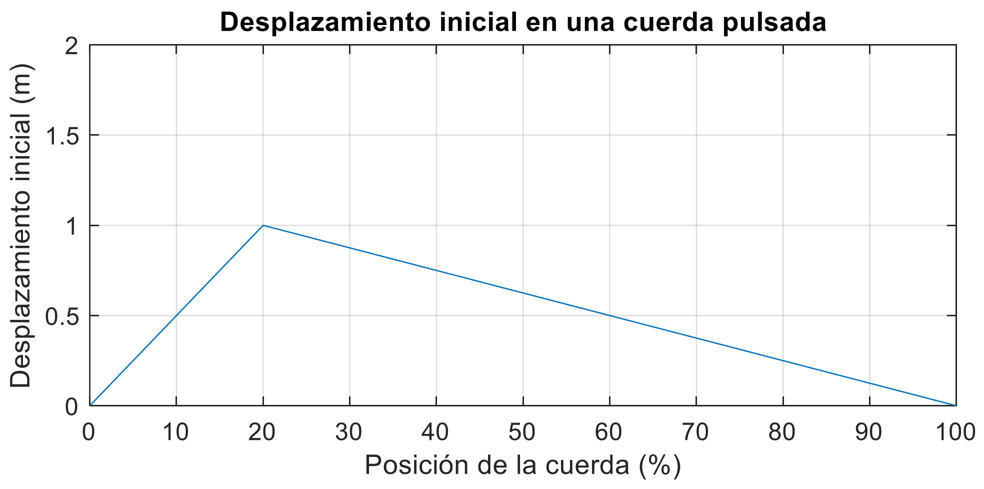


Figure14. Initial displacement in plucked string instruments.

In the case of the Harpejji, its execution is different: it is played by stepping on the strings on the frets, without pressing them, giving the strings a non-zero initial speed. If the moment in which the string comes into contact with the fret is considered as the initial condition, at that moment there is also a zero displacement in the string. This arousal is more like a percussion string instrument like the piano, which is reflected in the sound of the instrument.

In theFigure15An example of the initial velocity in a string struck with a hammer from below at a point at 30% of its length is included. Only the area that comes into contact with the hammer acquires speed at the initial instant. The initial displacement is zero throughout the chord.

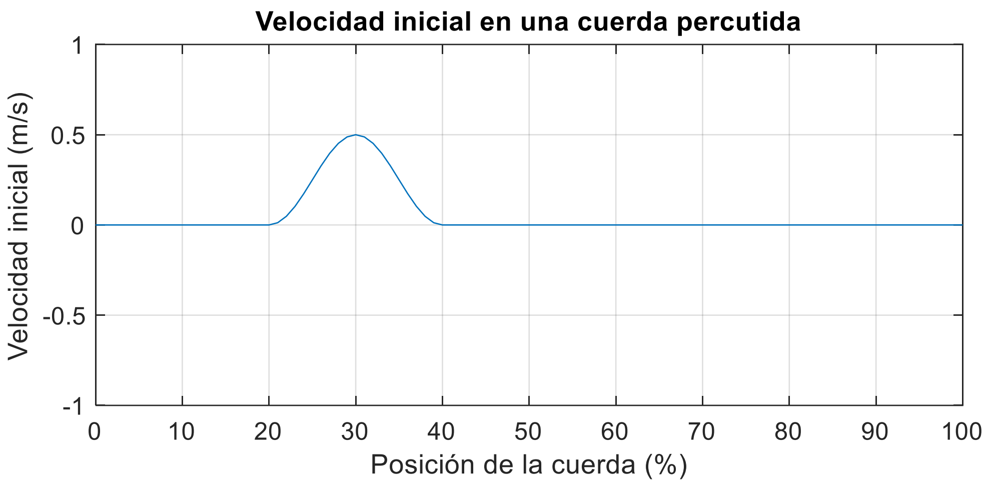


Figure15. Initial velocity in a string struck from below at the point at 30% of its length.

Unlike percussed string instruments, however, pressing the strings of the Harpejji imparts initial velocity to the entire string. Several possible excitations have been proposed that could reproduce that of the real instrument. For each of them, the spectrum they produce has been studied and compared with sounds recorded from a real Harpejji G16. All the proposed excitations are included below.

All the excitations with a maximum value equal to 1 m/s have been normalized to make it easier to work with them. The different excitations do not produce notes of the same amplitude, nor does the same excitation function produce notes of the same amplitude at different frequencies, so it is necessary to normalize their volume. This process is detailed in the section3.6.

3.5.1.Constant initial speed along the entire rope

As a first idea, it was proposed that the initial speed in the rope be constant throughout the rope, except for its fixed ends.

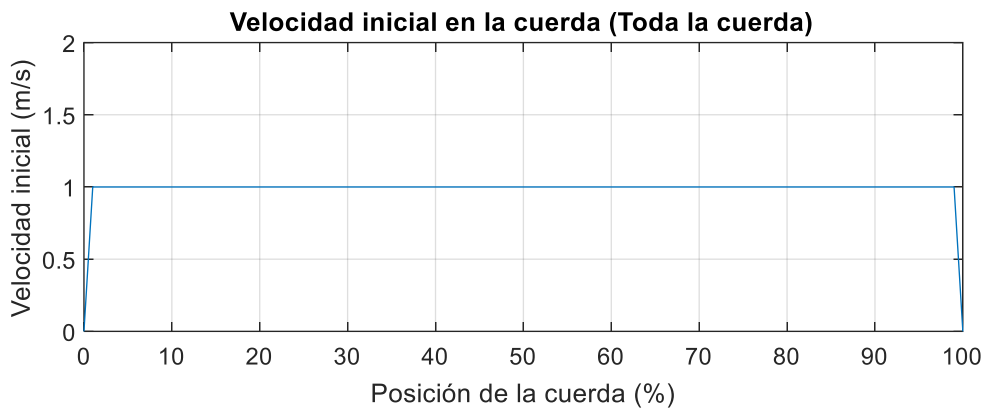


Figure16. Constant initial speed along the entire rope.

In order to verify the evolution of the spectrum in frequency, the spectrogram of the output signal of the synthesizer is made, excited with constant speed in the whole string and reproducing a note C3, with a fundamental frequency of 130.81 Hz. The spectrogram obtained is included in theFigure17.

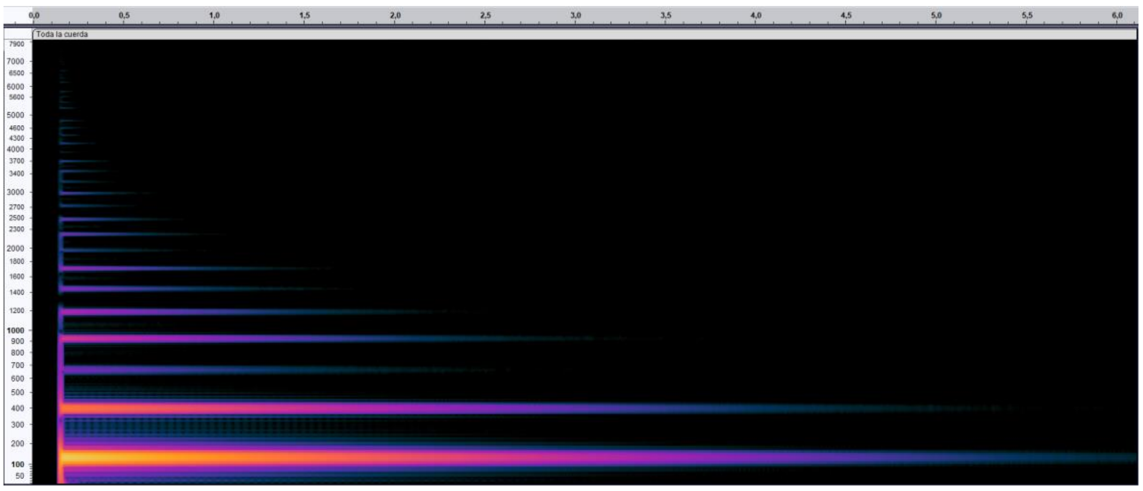


Figure17. Spectrogram of a C3 note played with equal excitation across the entire string.

This excitation causes a spectrum in which only the odd harmonics of the fundamental frequency appear.

3.5.2.Parable

An excitation of the string with an initial parabolic velocity along the string is then proposed, using the expression(44). The initial velocity in the rope has the form shown in theFigure18and generates the spectrogram of theFigure19.

$$v_0(x) = x \cdot (L - x) \tag{44}$$

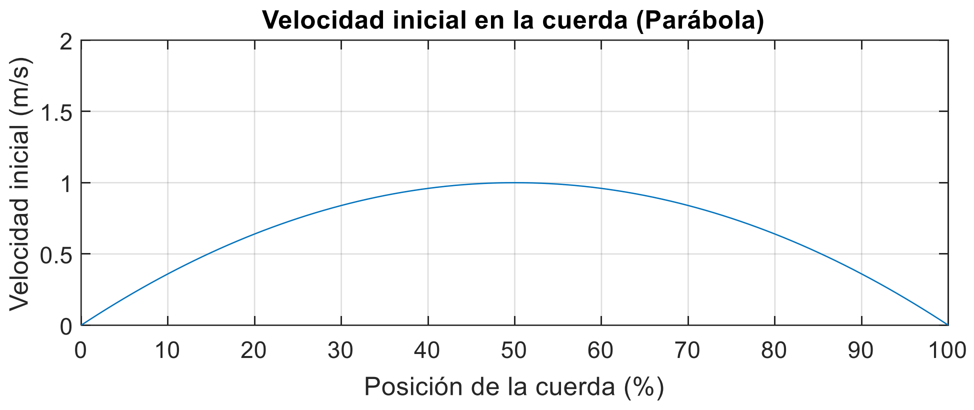


Figure18. Initial velocity in the shape of a parabola.

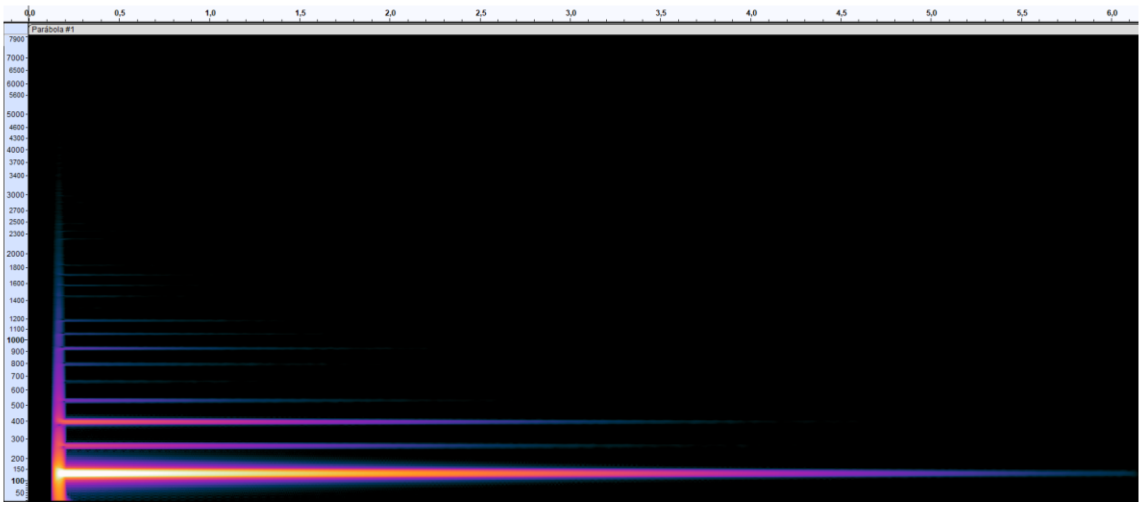


Figure19. Spectrogram of a C3 note with a parabolic excitation.

With this excitation both odd and even harmonics are generated, but the spectral content at high frequencies is very poor.

3.5.3.Ramp

Since the rope is stepped on from one of the ends, an excitation has been considered in which the maximum velocity occurs at the point closest to the end of the rope. As in the rest of the cases, the extremes have null initial velocity and displacement.

The expression of the equation is used(45)with a discontinuity at the last point of the rope to go from the maximum speed to the fixed end. The shape of the initial velocity in the rope can be seen in theFigure20.

$$v_0(x) = \frac{x}{L} \tag{45}$$

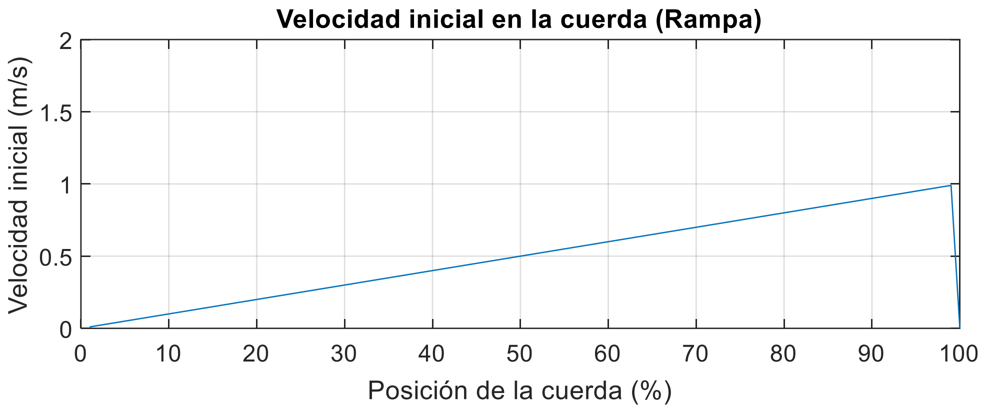


Figure20. Ramp-shaped starting speed.

The spectrogram of the instrument is made by reproducing a C3 note with the excitation in the form of a ramp, which is included in theFigure21.

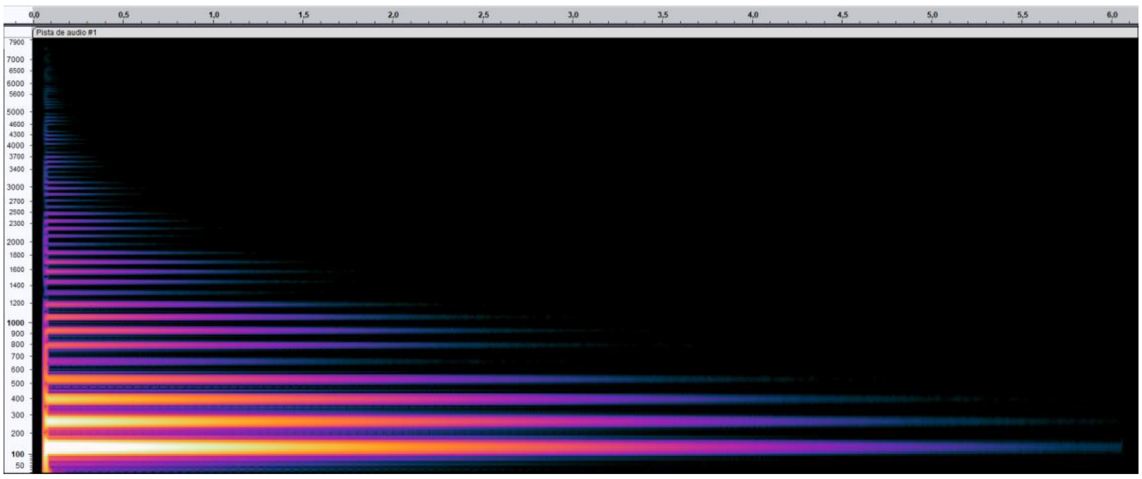


Figure21. Spectrogram of a C3 note reproduced with excitation in the form of a ramp.

With this excitation both odd and even harmonics are generated and a very rich spectral content, extending the harmonics up to almost 8 kHz.

3.5.4.shifted raised cosine

The ramp-shaped excitation considered in the previous section generates a large amount of harmonics, which can be reduced by using a seamless shape, with a smooth transition between the highest speed point and the fixed end.

A raised cosine function has been used in which the first half-period has been lengthened to the point of maximum speed and the second has shrunk to occupy from the point of maximum speed to the fixed end. To mathematically represent this waveform, the expression of the equation is used(47).

$$v_0(x) = \begin{cases} 0.5 - 0.5\cos\left(\dfrac{x}{x_{max}}\pi\right) & \text{si } x \leq x_{max} \\ 0.5 + 0.5\cos\left(\dfrac{x - x_{max}}{L - x_{max}}\pi\right) & \text{si } x > x_{max} \end{cases} \tag{46}$$

The figuresFigure22andFigure23they include the initial velocity on the string and the spectrogram produced by exciting the string with it playing a C3. The point of maximum speed, , is located at 90% of the rope.x_{max}

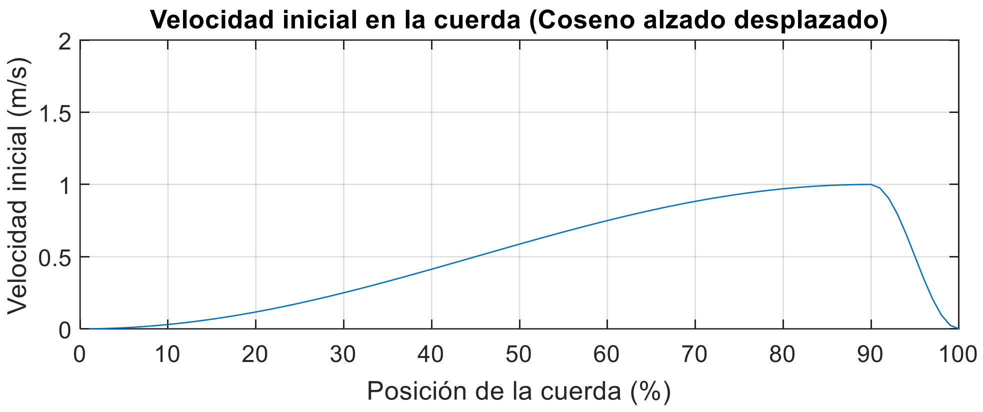


Figure22. Initial velocity in the form of a raised cosine displaced at 90% of the chord

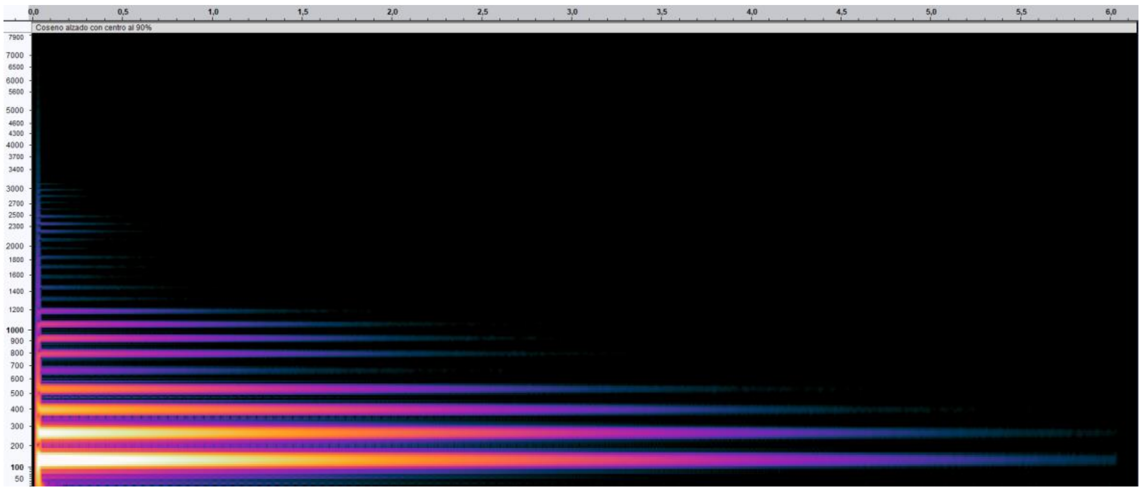


Figure23. Spectrogram of a C3 note with raised cosine excitation centered at 90% of the string.

With this excitation a good number of harmonics is achieved at high frequencies, both odd and even, and a sound relatively similar to that of Harpejji for the low and middle registers.

If the point of maximum velocity is moved towards the center of the string, the spectral content is reduced. in the figuresFigure24andFigure25the initial velocity in the rope and its spectrogram are shown respectively.

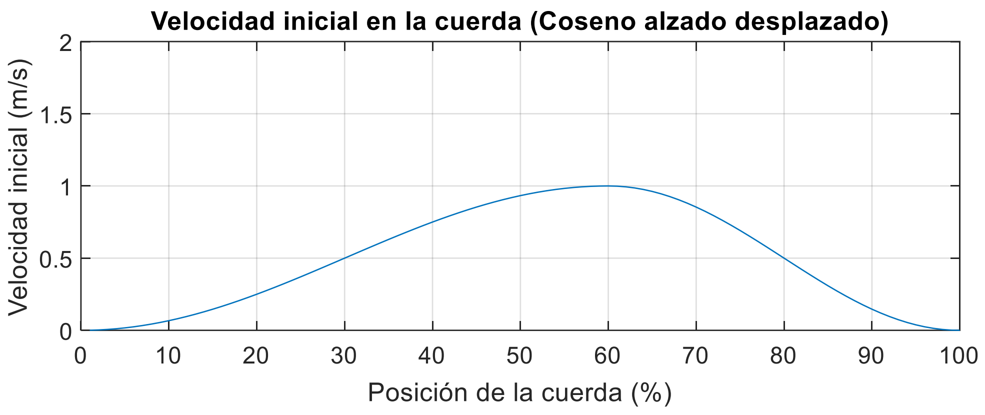


Figure24. Initial velocity in the form of a raised cosine displaced at 60% of the chord.

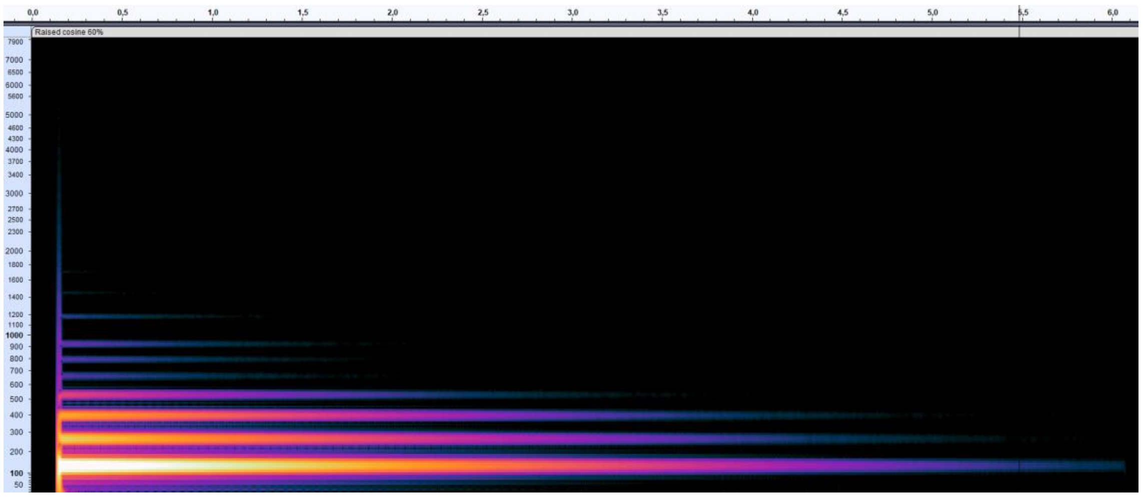


Figure25. Spectrogram of a C3 note with raised cosine excitation centered at 60% of the string.

3.5.5.Projectile trajectory function with friction

Subsequently, a function similar to that of the modified raised cosine of section3.5.4. but without the discontinuities that can be observed in theFigure22and that they would not occur on the real string.

For this, the function of the trajectory of a projectile with air resistance and Stokes friction is used, [20]. The values of mass, velocity and initial direction of the projectile are modified, as well as the friction of the air to achieve functions similar to those obtained using the raised cosine. The functions obtained are included in the figuresFigure26andFigure28and in the figuresFigure27andFigure29their respective spectrograms when playing a C3 note with this excitation.

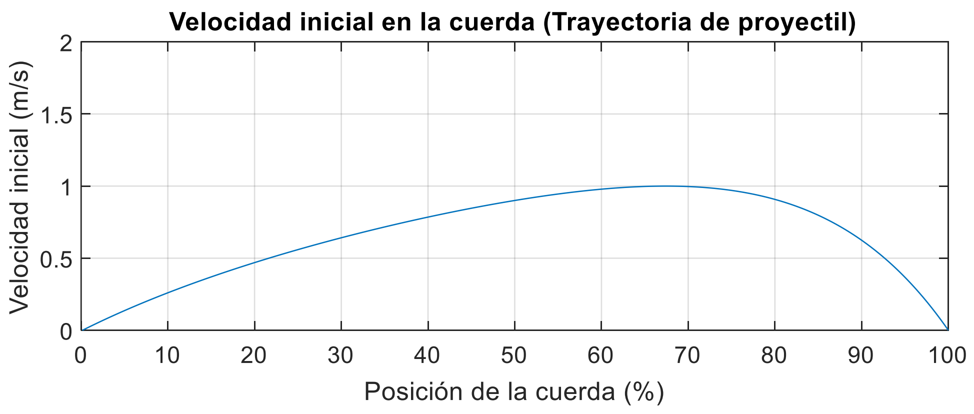


Figure26. Initial velocity with projectile trajectory shape 1.

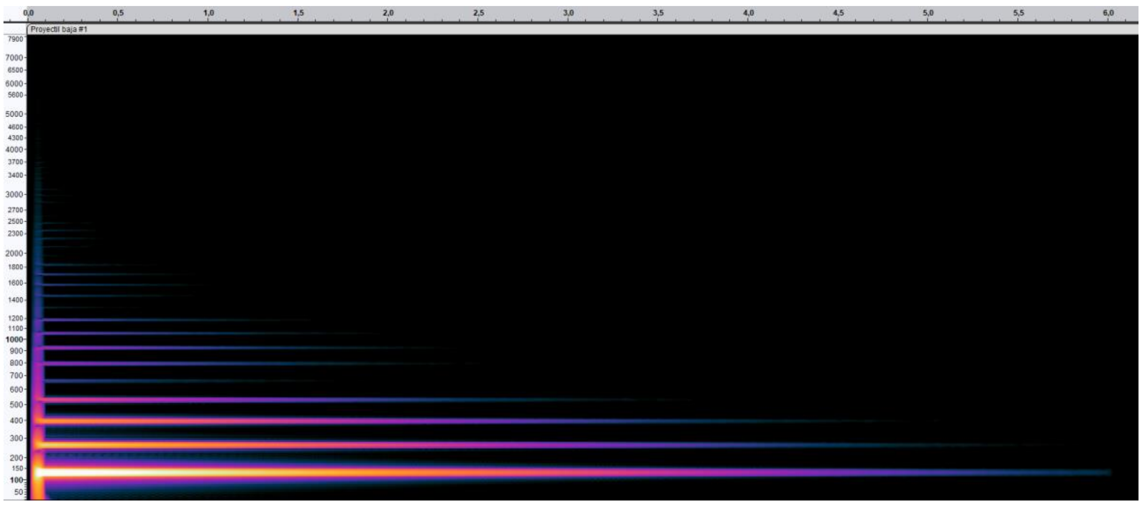


Figure27. Spectrogram of a C3 note with an excitation shaped like projectile trajectory 1.

In the first case, with the point of maximum speed close to 70% of the string, a spectrum with lower harmonic content at high frequency is obtained.

By moving the point of maximum velocity to 90% of the string, a spectrum richer in harmonics is achieved.

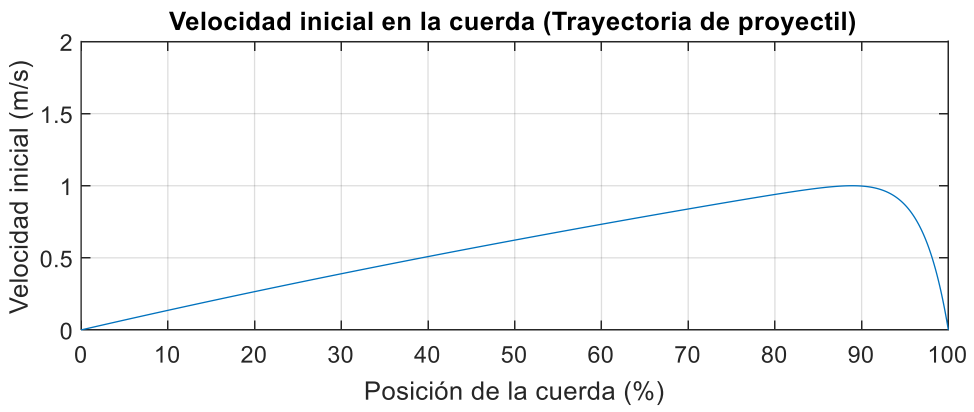


Figure28. Initial velocity shaped as projectile trajectory 2.

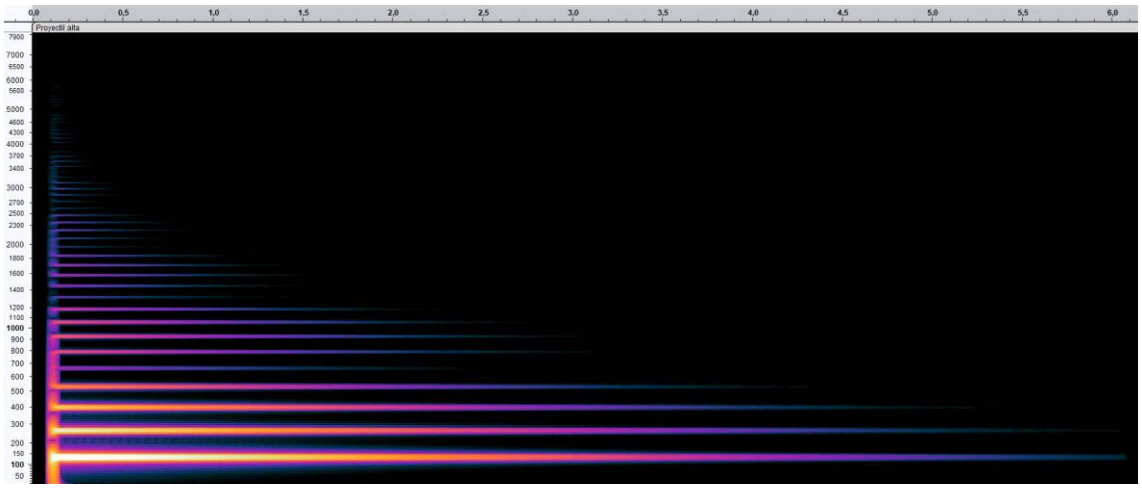


Figure29. Spectrogram of a C3 note with an excitation shaped like projectile trajectory 2.

In the final version of the plugin a weighted combination of the functions of the figures has been usedFigure26andFigure28according to the velocity of the received MIDI notes.

For those notes that are received with higher MIDI velocity, the function in figure 26 is used, which produces a note with greater spectral content and a more pronounced attack when the note begins to sound. For those with lower speed, the function of figure 24 is used, which produces fewer harmonics and a softer sound. For all intermediate cases a weighted sum of the two functions is used according to the received MIDI velocity.

This process is done in theCode2, where`vHigh`is the excitation of Fig. 26 and`vLow`, that of figure 24.

```
for (int x = 1; x < X; x++) {
    // Forma para velocity altas -> sonido más percusivo
    v0[x] =  velocity * vAlta;
    // forma para velocity baja -> sonido más suave
    v0[x] = v0[x] +  (1 - velocity) * vBaja;
}
```

Code2. Weighting of the two excitations of the string according to the MIDI velocity.

3.6.amplitude correction

The amplitude of the string vibration (and thus the volume of the plugin's audio output) varies as a function of the string length, the frequency of the oscillation, and the initial velocity of the string, and can be obtained using the Fourier method [21]:

$$b_n = \frac{2}{L\omega_n}\int_0^L v_0 \operatorname{sen}(k_n x)\, dx \tag{47}$$

To correct this deviation and ensure that all notes have the same volume, the integral over the entire string is calculated for the initial excitation used in each case; using for it discrete integration:

$$\int_0^L v_0 \operatorname{sen}(k_n x)\, dx \cong \sum_0^L v_0 \operatorname{sen}(k_n x)\, \Delta x \tag{48}$$

The expression for calculating the amplitude from the initial velocity at each discrete point on the string is as follows:

$$b_n = \frac{2}{L\omega_n} \sum_0^L v_0 \operatorname{sen}(k_n x)\, \Delta x \tag{49}$$

After calculating the amplitudeb_n, according to(49),the initial velocity is divided by this value so that all notes have the same volume. This amplitude normalization processcan be translated into C++ code as shown in theCode3.

```
// Se inicializa la integral con valor 0
float integral = 0;

// Se recorren todas las posiciones de la cuerda sumando la velocidad
inicial en ese punto ponderado por sen(kx) a la integral
for (int x = 0; x < X; x++) {
integral = integral + v0[x] * sin(kn * x * dx) * dx;
}

// Se calcula la amplitud de la vibración resultante de esa velocidad
inicial en la cuerda
b_n = (2 * integral / (L * 2 * float_Pi * frequency));

// Se normaliza la velocidad inicial en la cuerda
for (int x = 0; x < X; x++) {
v0[x] = v0[x] / b_n;
}
```

Code3. String vibration amplitude correction.

Once the amplitude is normalized, the initial velocity on the string is multiplied by the velocity of the received MIDI note that JUCE provides as a value of type float between 0 and 1.

3.7.Losses

From the decay time, , at any two frequencies, the parameters y can be calculated, which define the linear and frequency-dependent losses, respectively.$T_{60}\sigma_0\sigma_1$

For this we use the equations(50)and(51), taken from[4].

$$\sigma_0 = \frac{6\ln(10)}{\xi(\omega_2) - \xi(\omega_1)} \left(\frac{\xi(\omega_2)}{T_{60}(\omega_1)} - \frac{\xi(\omega_1)}{T_{60}(\omega_2)} \right) \tag{50}$$

$$\sigma_1 = \frac{6\ln(10)}{\xi(\omega_2) - \xi(\omega_1)}\left(-\frac{1}{T_{60}(\omega_1)} + \frac{1}{T_{60}(\omega_2)}\right) \quad (51)$$

The value of the is modified until a spectrogram similar to the one obtained from the real instrument is achieved. Specifically, one of 9 seconds at 200 Hz and one of 4 seconds at 10 kHz is used.$T_{60}T_{60}$

3.8.level detector

Since the movement of the string has a negative exponential amplitude, it never reaches zero until the precision of the variables used (float) is exceeded. As long as the voice playing the note has not finished playing it, it is not released so that a new note can be played on it.

To release the voice as soon as it is no longer audible, an RMS level detector has been implemented, which takes the string reading point as input and detects when the vibration falls below a certain threshold. The level detector has the form of the equation(52).

$$c_n^2 = \alpha x(n)^2 + (1-\alpha)c_{n-1}^2 \quad (52)$$

A parameter is chosen, such that the integration time of the level detector is 1 ms, which has been experimentally verified to achieve a fast detector response both when holding the note down and when releasing it.α

$$\alpha = \frac{1000}{1(ms)}\frac{1}{f_s} \quad (53)$$

A threshold of -90 dBFS has been chosen, which has been found to be sufficient so that clipping is not noticeable when you stop playing the note in that voice.

$$-90\ \mathrm{dBFS} = 20\log\left(\frac{\mathrm{valor}}{\mathrm{valor\ fondo\ de\ escala}}\right) \quad (54)$$

Since the full scale value in the working environment is equal to 1, it is obtained from the equation(54)a value of$10^{-4.5}$which has approached . The code referring to the level detector has been included in the5×10^{-5}Code4.

```
// Para cada muestra del buffer
for (int s = 0; s < synthBuffer.getNumSamples(); s++) {
      // Se calcula el desplazamiento de la cuerda y
      // se añade al buffer (omitido)
      {...}
      // Se introduce la muestra actual en el detector de
      // nivel RMS (c2n)
      c2n = alfa * powf(y[xRead], 2) + (1 - alfa) * c2n;

      // Si el nivel está por debajo del umbral (-90dbFS) se apaga la
      // nota
      if (c2n < 0.00005f) {
            {...}
            clearCurrentNote();
            return;
      }

      {...}
}
```

Code4. Level sensor.

3.9.MIDI input management

The VST plugin receives MIDI messages from the host DAW. The virtual instrument has been programmed to react only to Note On and Note Off messages, which contain information regarding the pitch and velocity of notes. All other MIDI messages (Aftertouch, pitch bend, program change, etc. [18]) are ignored.

When a Note On message is received, the synthesizer searches for a free voice and executes the function`startNote()`, which has been programmed to call the function`setInitialConditions()`, which establishes the characteristics and the initial velocity of the rope. To do this, the frequency (in hertz) is extracted from the MIDI message, using the function`getMidiNoteInHertz`() included in the JUCE library[22]and the velocity of the note (normalized between 0 and 1), which the class provides as an argument to the function.

```
void SynthVoice::startNote(int midiNoteNumber, float velocity,
juce::SynthesiserSound* sound, int currentPitchWheelPosition) {

      float frequency = getMidiNoteInHertz(midiNoteNumber);

      // Se comprueba que la nota está en el rango de notas que pueden
      // ejecutarse en el Harpejji (C2 - C6)
      if (frequency > 65.40f && frequency < 1047)
            setInitialConditions(velocity, frequency);
      else
            clearCurrentNote();
}
```

Code5. Management of MIDI messages.

3.10.tone control

A pitch control just like the one on the real instrument has been included in the plugin. Since no “real” Harpejji has been available and documentation exists in this regard, it has been assumed that the circuitry is identical to that of an electric guitar tone control. It normally consists of a first-order RC low-pass filter, made up of a capacitor and a variable resistor that regulates the cut-off frequency of the filter. The tone control schematic is included in theFigure30.

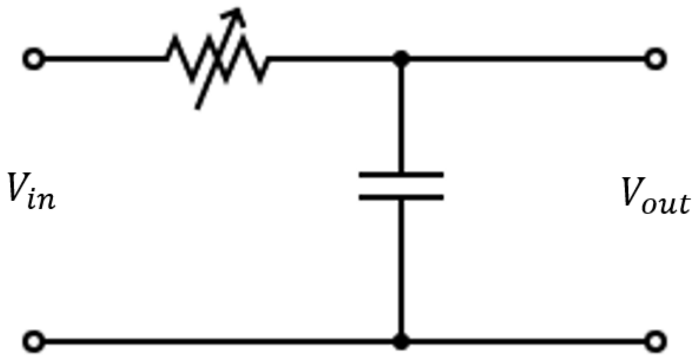


Figure30. First order low pass RC filter.

For its implementation in the program, a predefined IIR low-pass filter from the JUCE DSP library [23] has been used. The filter is initialized with a cutoff frequency of 20 kHz and the user is allowed to control this parameter from the graphical interface, being able to vary the cutoff frequency from 20 kHz to 20 Hz. This type of filter presents a frequency response like that of theFigure31, taken from [24], which produces a 3 dB per octave rolloff from the filter cutoff frequency.

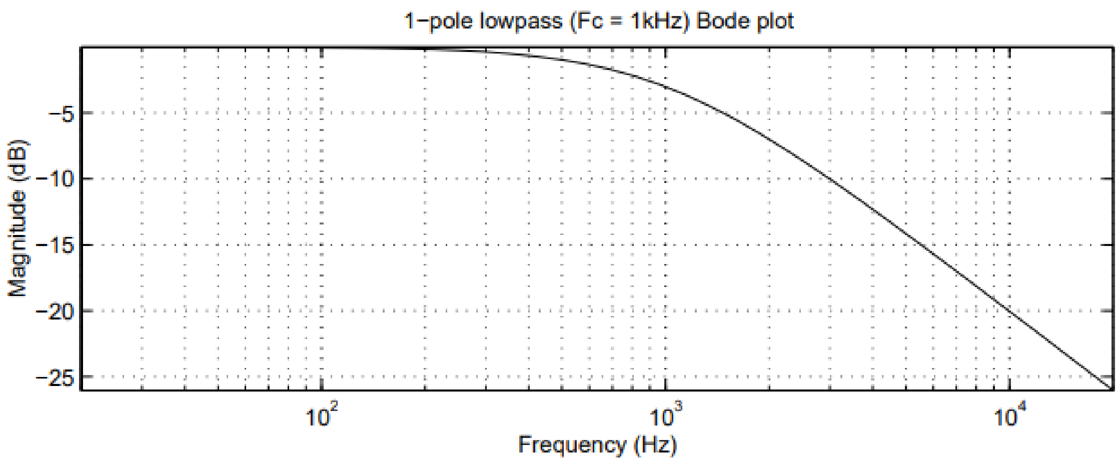


Figure31. Frequency response of the first order low pass filter.

3.11.gain control

The output signal of the filter is given a gain according to the value of the rotary control of the graphical interface. For this, a structure from the JUCE DSP library is also used, which allows a gain to be applied both in dB and in linear units to a sample or a complete audio block.

3.12.graphical user interface

A graphical interface has been designed that shows the user what notes and in what positions are being played at any moment and that allows the control of the instrument's parameters through four rotary virtual potentiometers that allow the user to control the tension of the strings, the time of decay, the cutoff frequency of the lowpass filter and the output gain of the plugin respectively. The plugin controls are displayed on theFigure32.

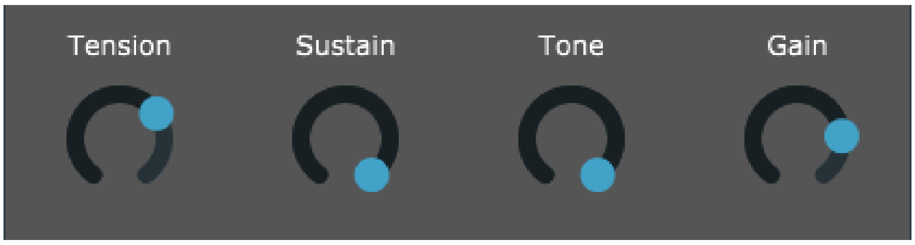


Figure32. Plugin interface rotary controls.

The four sliders are created, using Juce's Slider class, and drawn on top of the plugin window. The margins relative to the plugin area are established in each potentiometer so that when the plugin window is resized the size of the potentiometers is modified accordingly. Thus, each potentiometer occupies a sixth of the width of the window, both in height and width. Set the position of the first knob to be one-eighth of the window width minus one-half of the knob width, and the rest are shifted by an amount equal to the window width divided by 4. Code for this process has been included in theCode6.

```
void SynthAudioProcessorEditor::resized()
{
    const auto bounds = getLocalBounds();
    const auto sliderWidth = bounds.getWidth() / 6;
    const auto sliderHeight = bounds.getWidth() / 6;
    const auto sliderStartX = bounds.getWidth() /8 - (sliderWidth/2);
    const auto sliderStartY = 15 + 92 / 2 - (sliderHeight / 2);

    tensionSlider.setBounds(sliderStartX, sliderStartY, sliderWidth,
sliderHeight);
    sustainSlider.setBounds(sliderStartX + bounds.getWidth() / 4,
sliderStartY, sliderWidth, sliderHeight);
    toneSlider.setBounds(sliderStartX + 2 * bounds.getWidth() / 4,
sliderStartY, sliderWidth, sliderHeight);
    gainSlider.setBounds(sliderStartX + 3 * bounds.getWidth() / 4,
sliderStartY, sliderWidth, sliderHeight);
}
```

Code6. Rotary control positions.

The background image of the plugin has been designed using the GIMP software, so that it represents the surface of the Harpejji on which the notes are played. The distance between the frets has been represented to scale. For this, the formula for the distance has been obtained from [25],d,between fret number$n$and the nut. The generated image is included in theFigure33.

$$d = s \cdot (1 - e^{-kn}) \tag{55}$$

Where$s$is the scale length and$k = 0.05716$

The height of the image in pixels is taken as the scale length and the position of the 19 frets is calculated with the formula(55). The generated image of theFigure33it is inserted into the JUCE project, which allows it to be loaded as the background of the plugin.

On this surface are drawn, in execution, the 16 strings of the instrument, as well as the rotary controls for the four parameters as they appear in theFigure34, using the JUCE Path and Graphics classes ([26] and [27]), which allow drawing lines between two points in the plugin window.

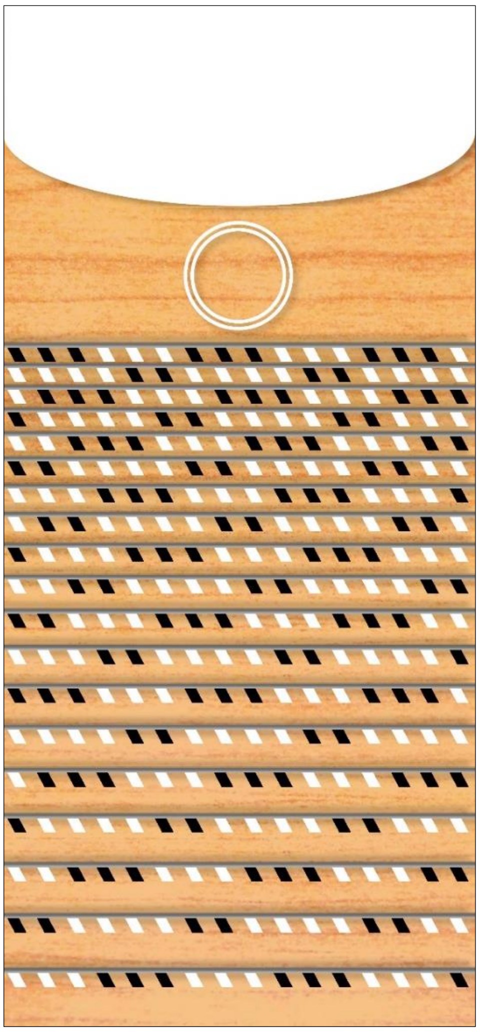


Figure33. Plugin background.

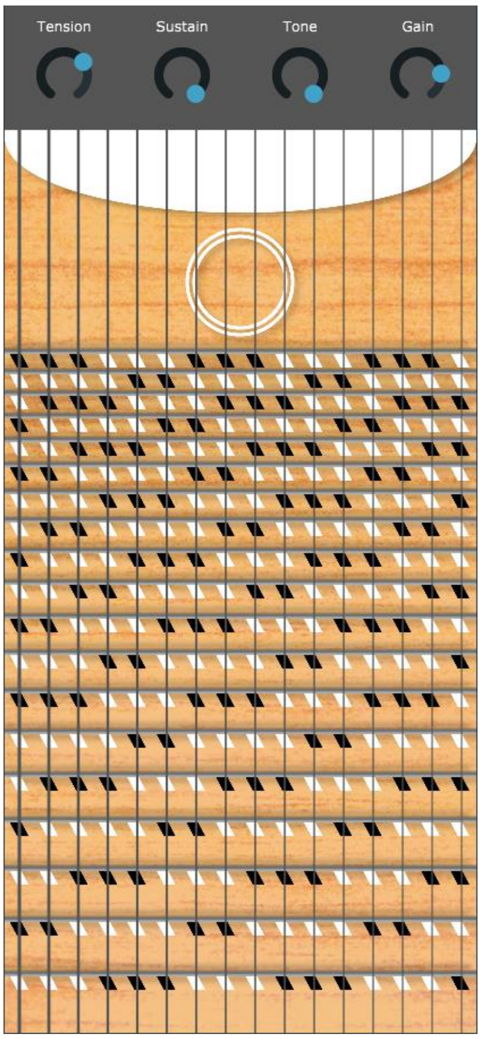


Figure34. Plugin user interface.

Pressing a note draws a black circle over the position on the instrument where you would place your finger on the actual instrument. Additionally, the vibration of the string is rendered which is passed as a vector to the PluginEditor class. The vibration of the string is scaled using again the equation(55)to represent it from the corresponding fret up to the top of the window.

It is included in theFigure35an image of the plugin playing various notes. The vibration of the strings is not reflected well, as it is a static image.

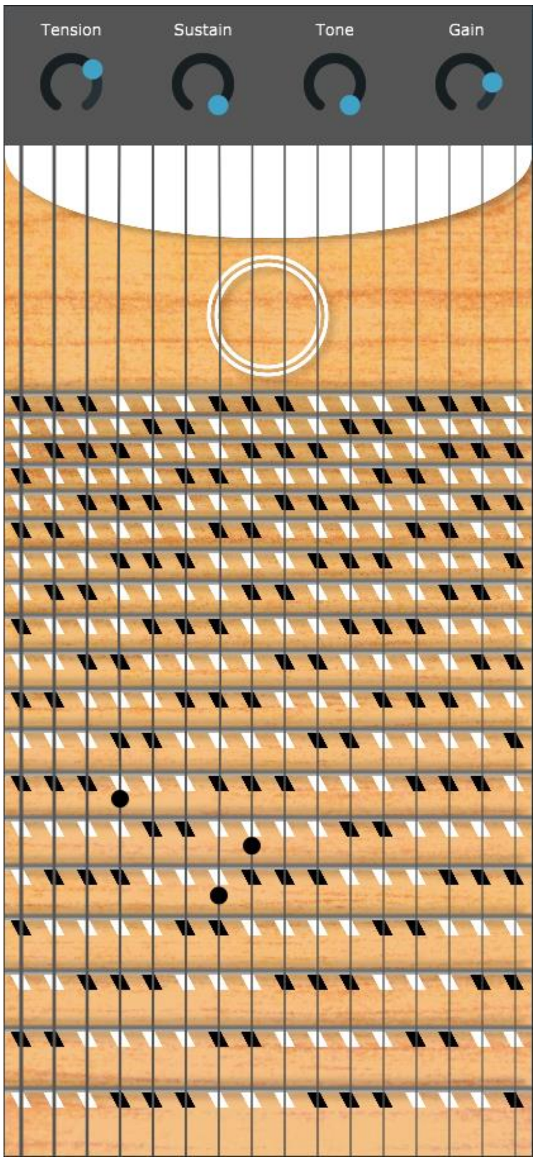


Figure35. Plugin interface playing various notes.

4. Results

In this chapter the results of the project are analyzed: the sound of the plugin is compared with that of the real instrument, the results obtained in terms of the operation and performance of the program are explained, and it is verified which of the objectives established for the project have been met.

4.1.Model validation

The maximum spectrum and the spectrogram of the sound of the plugin and of real Harpejji recordings are used to analyze the similarity achieved between the plugin and the real instrument, taken from [28]. in the figuresFigure36andFigure37Spectrum peaks are included when playing a C3 note on the real instrument and plug-in respectively.

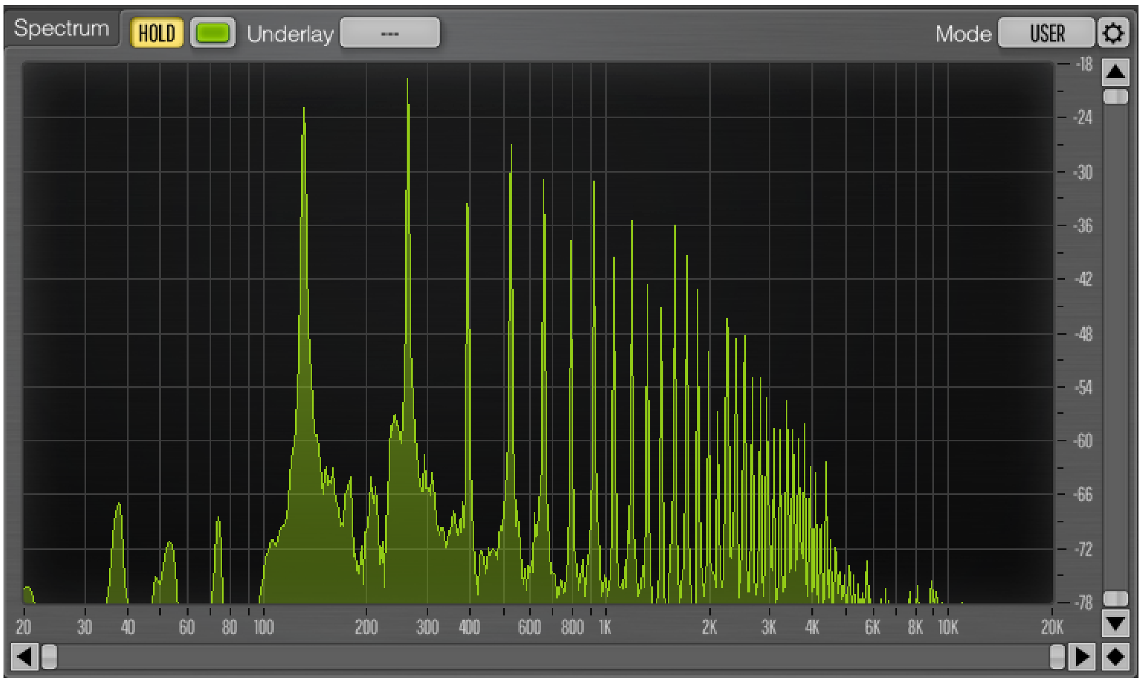


Figure36. Peak spectrum of a C3 note played on a real Harpejji.

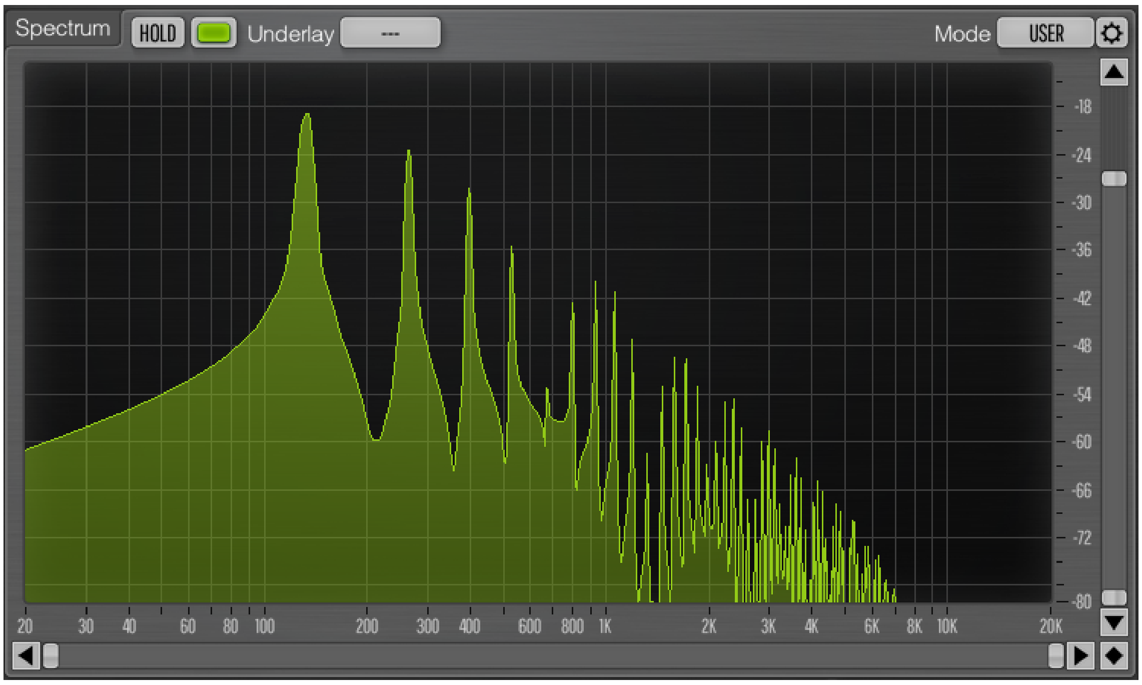


Figure37. Maximum spectrum of the plugin playing a C3 note.

It is observed that in general quite similarity is achieved in terms of harmonic density and the harmonics appear at exactly the same frequencies in the plugin and in the real instrument. Some frequencies appear attenuated in the plugin spectrum due to the reading point chosen on the string and the shape of the initial excitation used.

The spectrograms of the two sources are compared below. In this case playing a B3 note, since it is the longest isolated note that it has been possible to find; Also taken from [28]. Spectrograms are included in the figures.Figure38andFigure39.

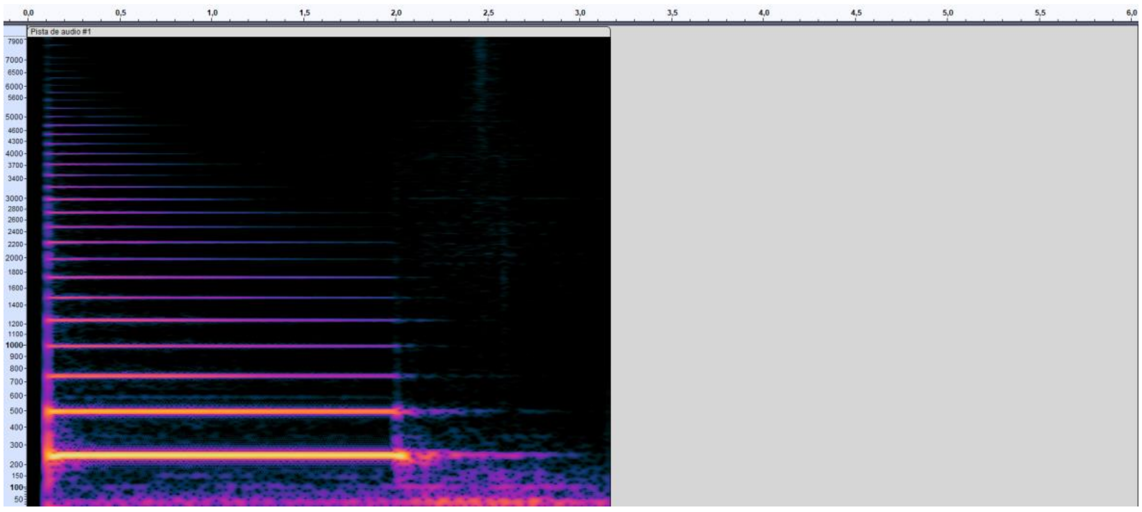


Figure38. Spectrogram of a B3 note played on the real instrument.

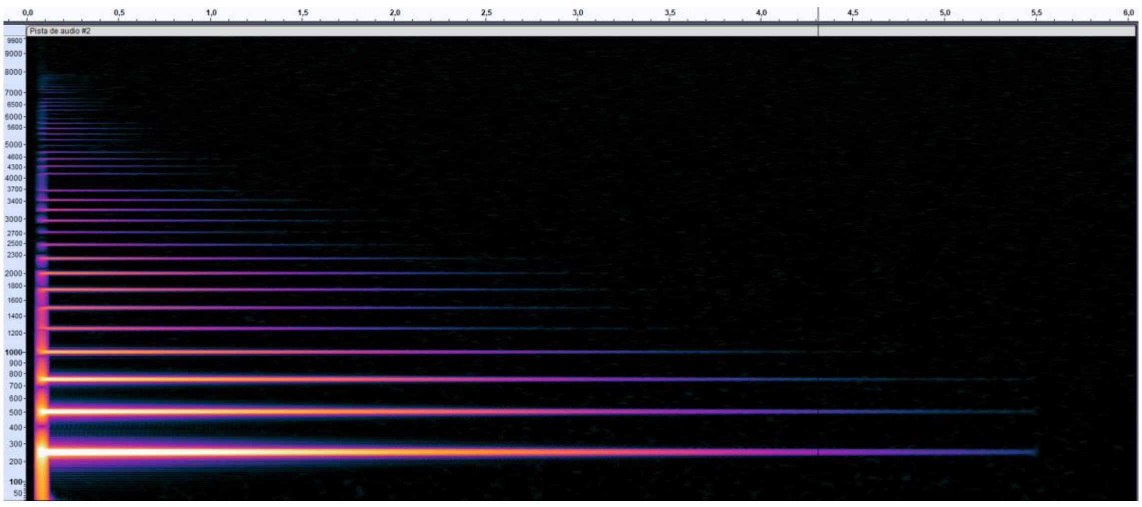


Figure39. Spectrogram of a B3 note played in the plugin.

The attenuation that appears in some of the harmonics is observed again. Both in the spectra and in the spectrograms it is observed that the harmonics at high frequencies appear with less amplitude in the virtual instrument than in the real one. The decay time closely approximates that of the real instrument at all frequencies.

The sound of the instrument is also verified by listening to it and it is compared with the sound of the real instrument, observing great similarity between the two.

4.2.Plugin operation

In the final version of the plugin, six voices have been used for the synthesizer, this number being the maximum number of voices possible without problems with clicks and audio cuts due to the high number of operations that the processor has to perform simultaneously.

Rotary controls do not clip or click the audio, and can be manipulated in real time while sound is playing in the plugin. The parameter controls relating to the physical modeling of the string have no effect on notes that are already playing, but rather have an effect on “spawning” a new string upon receipt of a new MIDI note.

The calculation of the stiffness term has been dispensed with in order to be able to introduce a greater number of voices without producing audio errors. Tests have been carried out and removing this term has no impact on the sound in most cases, except for those extreme cases, with very low tension or very high linear mass of the string.

The graphical interface shows the vibration of the strings in real time and indicates with a dot those positions in which the string is being pressed.

The audio output is adjustable to avoid clipping when passing the signal from the plugin to the DAW.

4.3.Evaluation of the objectives

Once the development of the project is completed, it is considered which of those objectives and design restrictions detailed in1.5has been satisfied.

Although it has been necessary to have access to a computer with the MacOS operating system and one with Windows to generate the VST3 plugin compatible with each system, it has been possible for the plugin to work identically on both systems.

As detailed in chapter4.1, it has been possible to emulate the sound of the Harpejji, achieving a good similarity, although there are improvements that can still be included in the plugin to get its sound even closer to the real instrument and that are explained in the section6.2.

The plugin responds to MIDI input messages provided by the DAW, and plays the corresponding notes in real time, with no user noticeable latency and no audio dropouts if the buffer size of the computer's audio adapter is set correctly. The plugin's output signal is monophonic as specified in the design constraints.

Depending on the speed with which the MIDI notes are played or entered into the plugin, the instrument varies both the output volume and the harmonic content of the sounds that are played at its output.

The user is allowed to control the physical parameters of the instrument's strings (string tension and the decay time of the strings, related to the damping of the air and the string itself) as well as controlling the output volume of the plugin and the instrument tone control.

Regarding the polyphony of the instrument, a number of six voices has been chosen as the maximum number of voices without finding problems of audio cuts caused by the high number of operations that do not give time to execute in the time between reading a frame to the output of the plugin and the following.

In order to reduce the operating load and thus increase the number of voices that can be processed, the string stiffness term has been eliminated from the wave equation, which is explained in detail in the sections2.5and3.4.6. This term has negligible impact on the sound of the instrument for "normal" cases, within the actual physical characteristics of the strings, but it was included to add realism to extreme cases, where the string behaves more like a string. rigid bar. In the final version of the plugin it has been necessary to remove it.

Therefore, it has been possible to satisfy all the objectives established in the preliminary project phase. Even so, there are improvements that can be made to the plugin to achieve a better sound, operation and interface and that are explained in the section6.2.

5. Budget

Included in this section are the expenses derived from the development of the project divided between labor and materials.

5.1.Labour

Concept	Unit of measurement	Quantity of Units	Unit price	Amount
Engineering	Hours	300	€20.00	€6,000.00
			Total	€6,000.00

Total labor: SIX THOUSAND EUROS

5.2.Material

Concept	Price	Residual value	useful life (years)	Amortization
Computer	€750.00	€100.00	10	€65.00
peripherals	€250.00	€0.00	5	€50.00
MIDI keyboard	€100.00	€25.00	5	€15.00
Cubase DAW License	€300.00	€0.00	10	€30.00
Non-Commercial JUCE License	€ -	€ -	-	€ -
Microsoft Visual Studio License	€ -	€ -	-	€ -
			Total amortization	€160.00

Concept	Quantity of Units	Unit price	Amount
Harpejji string set	1	€50.00	€50.00
Precision scale	1	€30.00	€20.00
		total purchases	€80.00

Total material resources: TWO HUNDRED FORTY EUROS

5.3.Total

Concept	Amount
Labour	€6,000.00
Material	€250.00
Total	€6,250.00

Total: SIX THOUSAND TWO HUNDRED FORTY EUROS

6. conclusions

This chapter presents the conclusions obtained after the development of the project, as well as the difficulties encountered and the future work that could be done to improve the synthesizer. The contributions made by the author are also clarified.

6.1.Difficulties

This section mentions the difficulties that have been encountered both in the draft phase and in the development of the project and how they have been overcome.

To faithfully model the instrument it is necessary to know the physical characteristics. In the first instance, the instrument manufacturer was contacted, who was asked for information about the scale length of the instrument, as well as the gauge of the strings it used. After explaining the objective of the project and ensuring that it is an academic and non-commercial project, the manufacturer refused to provide any of this data.

Given the circumstances, it has been necessary to estimate the scale length of the instrument at 27” or 68.58 cm, based on images of the instrument. Regarding the gauge of the strings, they were obtained from a publication on the social network profile of a Harpejji performer [29], which is included in theFigure40.

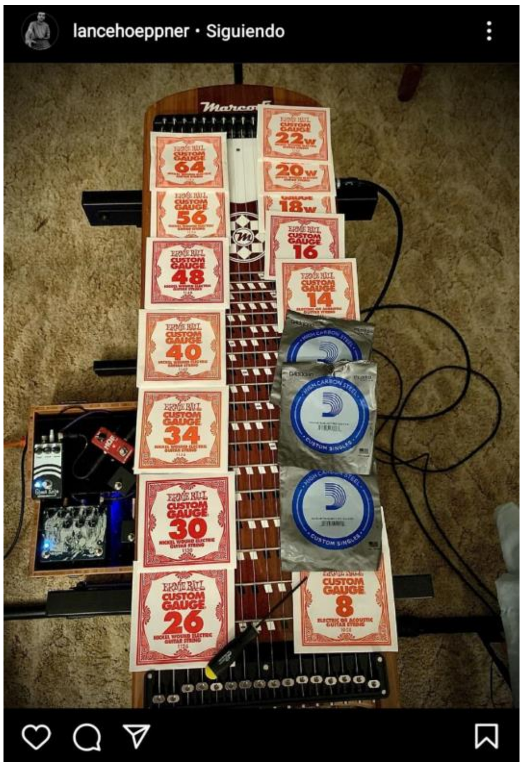


Figure40. Post on the social network Instagram showing a complete set of Harpejji strings.

After obtaining the information on the gauge of the strings, an order was placed for individual strings of the corresponding gauges on which the measurements included in the specification were made.Board1.

Another difficulty that has been encountered is related to the shape of the velocity on the string for this type of instrument. Although there is extensive literature on excitation for plucked, struck, or bowed string instruments, the Harpejji cannot be classified into any of these categories. The method of execution, pressing the strings directly on the frets, without pressing them, causes an initial excitation in the string in which the initial displacement is zero, but there is non-zero initial velocity throughout the string.

It has been necessary, therefore, to infer the shape of the initial velocity in the string, trying several excitations until finding one that produces a sound and a spectrogram similar to that of the real instrument.

The last difficulty that has been raised is related to the programming language used, C++. Programming in C, Java and MATLAB is taught during the degree. C++ has many similarities with C, but it has been necessary to familiarize yourself with the way of working in C++, as well as with the methods and structures of the JUCE library.

In order to solve this difficulty, the knowledge obtained from the subject Audio Engineering III has been very useful; in which we have worked with audio processors in real time, using MATLAB. Mention should also be made in relation to this difficulty of The Audio Programmer, on YouTube [30] and his series of videos explaining how to create a basic synth using JUCE, as well as the JUCE example projects [31], which helped very early in plugin development.

6.2.Future work

As mentioned above there are many aspects of the plugin where improvements can be made. Some of them are proposed below.

- String excitation: Since the initial velocity on the string has to be inferred, it is to be expected that there will be differences between the proposed initial velocity function and the actual excitation that occurs when strumming the Harpejji strings. If you had access to the real instrument you could measure this excitation and include it in the plugin so that the sound produced would be even more similar to the real instrument.

- Decay times according to frequency: It has also not been possible to find a complete isolated Harpejji note from which the decay times at all frequencies can be obtained. With full note recordings of the real instrument one could set the decay times faithfully as they occur on the real instrument.

- *Slides*: A common technique when playing the Harpejji consists of making slides or slides between two notes on the same string. There is no easy way to implement this behavior using physical modeling as this would require modifying the length of the string on which a note is being played while calculating the offset on it in subsequent samples. Changing the length of the string would lead to stability issues. One option to implement this technique could be through the use of a pitch-bend effect, which modifies the pitch of the input sound.

- Piezo Pickup Transfer Function: On the Harpejji, the vibration of the string is picked up by a piezo pickup located at one end of the string. To faithfully reproduce the sound of the instrument, it would be necessary to filter the output signal of the plugin's strings using the frequency response of the piezoelectric pickups.

- Improved algorithm to decide which string/fret a note is played on: A very simple algorithm has been used in the final version of the plugin to determine which position a note is played on. This affects both the sound, since each string has different characteristics, and the graphical interface that represents in which position the note is being played. With the current algorithm it is possible to play two different notes on the same string, even though only one of them is represented in the graphical interface. A more sophisticated algorithm for deciding which position to play the input note would be a simple plugin enhancement that hasn't been implemented due to time constraints.
- Improved plugin stability: some bugs related to the graphical interface are known to cause the plugin to crash. It has not been possible to fix them and it is an improvement that should be introduced to terminate the plugin.

6.3.contributions

It has been possible to implement a virtual Harpejji instrument using synthesis by physical modeling, which is a product that does not exist on the market. If the improvements mentioned in6.2their sale could be considered as a commercial product.

It has been necessary to fully infer the excitation of the string based on the playing method, since there is no literature on the matter of synthesis by physical modeling of this instrument or of others with a similar playing technique.

Although there are other examples in the bibliography, the code related to the solution of the wave equation by means of finite differences is original.

7. References

[1]"Harpejji Model Comparison," Marcodi Musical Products. https://www.marcodi.com/pages/harpejji-model-comparison (accessed April 30, 2022).

[2]«Harpejji»,*Wikipedia, the free encyclopedia*. August 27, 2020. Accessed: April 30, 2022. [Online]. Available at: https://es.wikipedia.org/w/index.php?title=Harpejji&oldid=128799314

[3]"Additive synthesis", Wikipedia, the free encyclopedia. January 6, 2020. Accessed: May 24, 2022. [Online]. Available at: https://es.wikipedia.org/w/index.php?title=S%C3%ADntesis_aditiva&oldid=12255502 9

[4] SD Bilbao, Numerical sound synthesis: finite difference schemes and simulation in musical acoustics. Chichester: Wiley, 2009.

[5]«Subtractive synthesis», Wikipedia, the free encyclopedia. February 19, 2020. Accessed: May 24, 2022. [Online]. Available at: https://es.wikipedia.org/w/index.php?title=S%C3%ADntesis_substraactiva&oldid=123 690639

[6]«Synthesis by frequency modulation», Wikipedia, the free encyclopedia. August 14, 2021. Accessed: May 24, 2022. [Online]. Available at: https://es.wikipedia.org/w/index.php?title=S%C3%ADntesis_por_modulaci%C3%B3n_de_frecuencias&oldid=137669380

[7]bryc, "Algorithms.md," Gist. https://gist.github.com/bryc/e997954473940ad97a825da4e7a496fa (accessed May 24, 2022).

[8]"JonDent - Exploring Electronic Music: Yamaha DX7 - Operators & Algorithms". https://djjondent.blogspot.com/2019/10/yamaha-dx7-algorithms.html (accessed May 24, 2022).

[9]«Synthesis by wave table»,*Wikipedia, the free encyclopedia*. September 13, 2021. Accessed: May 24, 2022. [Online]. Available at: https://es.wikipedia.org/w/index.php?title=S%C3%ADntesis_mediante_tabla_de_ondas &oldid=138322253

[10]White Noises,*Granular Synthesis EXPLAINED*, (June 1, 2018). Accessed: May 24, 2022. [Online Video]. Available at: https://www.youtube.com/watch?v=ftDLRYnRYZQ

[eleven]«Granular synthesis», Wikipedia, the free encyclopedia. January 6, 2022. Accessed: May 24, 2022. [Online]. Available at: https://es.wikipedia.org/w/index.php?title=S%C3%ADntesis_granular&oldid=1407729 65

[12]«Synthesis by physical modeling», Wikipedia, the free encyclopedia. March 3, 2022. Accessed: May 24, 2022. [Online]. Available at:

https://es.wikipedia.org/w/index.php?title=S%C3%ADntesis_por_modelado_f%C3%ADsico&oldid=142040193

[13]"Digital Waveguide Modeling Elements". https://ccrma.stanford.edu/~jos/pasp/Digital_Waveguide_Modeling_Elements.html (accessed May 24, 2022).

[14]«AAS- Tech Talk - Physical modeling». https://www.applied-acoustics.com/techtalk/physicalmodeling/ (accessed May 24, 2022).

[fifteen]NH Fletcher and TD Rossing, The Physics of Musical Instruments. New York, NY: Springer New York, 1998. doi: 10.1007/978-0-387-21603-4.

[16]"Young's Modulus", Wikipedia, the free encyclopedia. September 26, 2021. Accessed: May 25, 2022. [Online]. Available at: https://es.wikipedia.org/w/index.php?title=M%C3%B3dulo_de_Young&oldid=138590261

[17]«VST (Virtual Studio Technology)», ingeniatic. http://ingeniatic.euitt.upm.es/index.php/tecnologias/item/659-vst-virtual-studio-technology (accessed May 25, 2022).

[18]«JUCE: Synthesizer Class Reference». https://docs.juce.com/master/classSynthesiser.html (accessed May 25, 2022).

[19]J. Walsh, "Finite-Difference and Finite-Element Methods of Approximation," Proc. R. Soc. London. Be. Math. Phys. Sci., vol. 323, no. 1553, p. 155-165, 1971.

[twenty]"Projectile motion," Wikipedia. May 31, 2022. Accessed: June 3, 2022. [Online]. Available at: https://en.wikipedia.org/w/index.php?title=Projectile_motion&oldid=1090776713

[twenty-one]Physics Learning With Dr. Shaw,*Solution of wave equation for struck string | Struck String | Vibration of String | Part 4*, (June 29, 2020). Accessed: June 6, 2022. [Online Video]. Available at: https://www.youtube.com/watch?v=FfIcsCAyg3E

[22]«Essentials of the MIDI protocol». https://ccrma.stanford.edu/~craig/articles/linuxmidi/misc/essenmidi.html (accessed May 29, 2022).

[23]"JUCE: dsp::IIR::Coefficients< NumericType > Struct Template Reference". https://docs.juce.com/master/structdsp_1_1IIR_1_1Coefficients.html#a70e856c050b1fa38659b1b67469f567a (accessed May 29, 2022).

[24]Max Kamenetsky, "Filtered Audio Demo." Accessed: May 29, 2022. [Online]. Available at: https://web.stanford.edu/~boyd/ee102/conv_demo.pdf

[25]RM Mottola, "Liutaio Mottola Lutherie Information Website," Liutaio Mottola Lutherie Information Website. https://www.liutaiomottola.com/ (accessed May 31, 2022).

[26]«JUCE: Path Class Reference». https://docs.juce.com/master/classPath.html (accessed May 31, 2022).

[27]«JUCE: Graphics Class Reference». https://docs.juce.com/master/classGraphics.html (accessed May 31, 2022).

[28]Harpejji by Marcodi, Harpejji Playing Basics, (November 17, 2020). Accessed: June 5, 2022. [Online Video]. Available at: https://www.youtube.com/watch?v=AUnvq2zRU6k

[29]«Lance Hoeppner on Instagram: "Getting ready for a few shows this month. All cleaned up with new strings! #harpejjig16 #harpejji #fortwaynemusicscene #fortwaynemusic…"», Instagram. https://www.instagram.com/p/CP3VCEJLCbh/ (accessed June 8, 2022).

[30]The Audio Programmer,*Let's Build a Synth with Juice Part 0 - Oscillator*, (December 3, 2020). Accessed: March 26, 2022. [Online Video]. Available at: https://www.youtube.com/watch?v=ADG6Rsd3ekg

[31]"JUCE: Tutorial: Build a MIDI synthesiser". https://docs.juce.com/master/tutorial_synth_using_midi_input.html (accessed March 26, 2022).

8.Bibliography

[32]JUCE, Martin Shuppius - Physical modeling of guitar strings (ADC'17), (November 20, 2017). Accessed: March 26, 2022. [Online Video]. Available at: https://www.youtube.com/watch?v=sxt5rxF_PdI

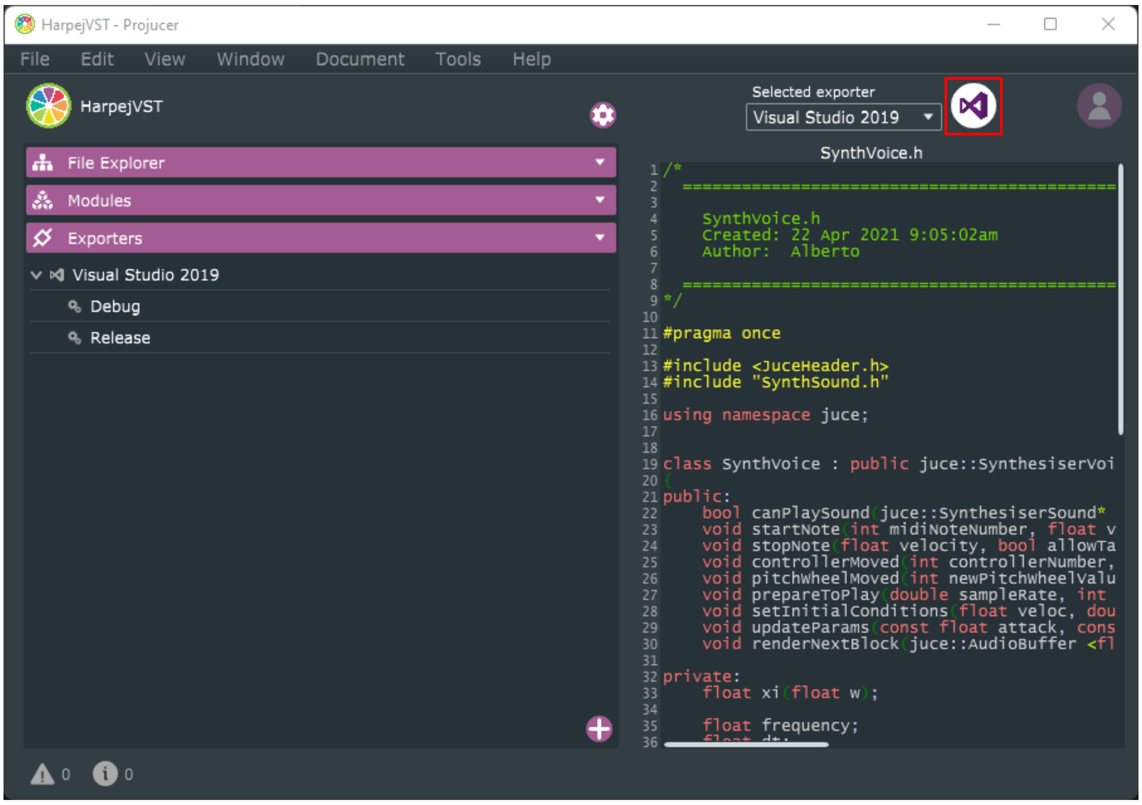
[33]E. Hayes, BatSynth. 2022. Accessed: March 26, 2022. [Online]. Available at: https://github.com/Emmet-Hayes/BatSynth

[3. 4]JWS Rayleigh, The theory of sound. Vol. 1, Rep. of 2. ed. rev. and enl.1894., vol. 1. New York: Dover Publishing, 1969.

Addendum: Installation Guide

To install the plugin on your computer, simply copy the HarpejVST.vst3 file from the Mac or Windows folder (depending on the operating system of the target computer) to the VST3 folder on your computer. By default it is located in C:\Program Files\Common Files\VST3 in the case of Windows systems and in Library/Audio/Plug-ins/VST3 in MacOs.

To make any changes to the plugin code, it is necessary to have installed both the JUCE editor, Projucer and a development environment (Usually Visual Studio on Windows and XCode on MacOs). Opening the Synth.jucer file with Projucer and then clicking on the Visual Studio or Xcode prompt (Save and open on IDE) opens the project in the chosen development environment from which it can be compiled into a new . vst3 file that can be placed in the VST3 folder on your computer so that the DAW automatically recognizes it.



```
/*
  ==========================================

    SynthVoice.h
    Created: 22 Apr 2021 9:05:02am
    Author:  Alberto

  ==========================================
*/

#pragma once

#include <JuceHeader.h>
#include "SynthSound.h"

using namespace juce;


class SynthVoice : public juce::SynthesiserVoi
{
public:
    bool canPlaySound(juce::SynthesiserSound*
    void startNote(int midiNoteNumber, float v
    void stopNote(float velocity, bool allowTa
    void controllerMoved(int controllerNumber,
    void pitchWheelMoved(int newPitchWheelValu
    void prepareToPlay(double sampleRate, int
    void setInitialConditions(float veloc, dou
    void updateParams(const float attack, cons
    void renderNextBlock(juce::AudioBuffer <fl

private:
    float xi(float w);

    float frequency;
```

Figure41. Projucer window with the project open. Circled in red the button to open the project in the Visual Studio development environment.

It must be taken into account that the .vst3 files generated in Xcode from MacOS will only work on computers with this same operating system. The same goes for Visual Studio and Windows.