Koto, Out of the Box

Marco Fiorini
mfiori21@student.aau.dk
Aalborg University Copenhagen
Sound and Music Computing, 8th Semester
Physical Modelling

***Abstract*—The Koto is an ancient stringed japanese instrument present at Musikmuseet, Danish Music Museum in Copenhagen. Being collected behind a glass cabinet, the instrument cannot be touched, played or heard. In this research I aimed at recreating a realistic digital application of the instrument through physical modelling synthesis. A digital waveguide model has been implemented in Faust and ported to Max/MSP. A custom interface recreating the visual setting of the instrument has been designed in TouchOSC and loaded on an iPad. By pressing the buttons, representing the virtual instrument strings, on the layout of the iPad's interface, OSC messages are sent to the main patch in Max/MSP and routed to the synthesis model parameters.**

I. Introduction

Physical modeling has quite a long history in the field of sound synthesis. It is based on the idea that when the vibrating structure is simulated in exactly the right way, the sound produced by that model is identical with the sound of the corresponding physical object. Thus, physical modelling of musical instruments simply means that the physical structure of a musical instrument is being modelled with mathematical and physical formulas realized with a computer [1]. A fundamental difference between the physical modelling approach and other synthesis techniques is that the former tries to imitate the properties of the sound source (based on physical reality and controlled by physical parameters, with a specific model for each instrument and synthesis independent from analysis), while the latter focus on the properties of the sound signal heard by the listener (based on perceptual reality and controlled by perceptual parameters, with a general model for all instruments and synthesis starting from simple unit generators).

Over the years, many different techniques have been proposed [2], such as digital waveguides [3], mass-interaction models [4], modal synthesis [5] [6] or finite difference schemes [7]. Even though sometimes more computationally demanding and difficult to control than other synthesis methods [8], physical modeling techniques offer many advantages. Indeed, creating a model of a vibrating system provides full control of its properties and, as a consequence, the output sound. These approaches theoretically allow us to synthesize natural and realistic sounds, tunable in every detail.

Therefore, as *"there are no theoretical limitations to the performance of a computer as a source of musical sounds, in contrast to the performance of ordinary instruments"* [9], the concrete applications of physical modellings techniques may range between a wide number of usages, including:

- Understanding sound production mechanism of musical instruments and everyday sounding objects (for acousticians),
- Developing algorithms which simulate in real-time sonorities created by such instruments (for computer scientists),
- Using the models to create sound effects and/or sonorities which do not exist in real-world (for sound designers and composers).

Fig. 1. The koto is a Japanese plucked half-tube zither instrument. The tube zither is a stringed musical instrument in which a tube functions both as an instrument's neck and its soundbox. As the neck, it holds strings and allows them to vibrate. Koto are roughly 180 centimetres in length, and made from Paulownia wood. The most common type uses 13 strings strung over movable bridges used for tuning, different pieces possibly requiring different tuning.

The presented project has the purpose of simulating an ancient japanese instrument, namely the Koto (see Figure 1) using physical modelling techniques. The instrument is collected in Musikmuseet, Danish Music Museum in Copenhagen [10] but being behind a glass cabinet it cannot be touched, played or heard. Thus, my implementation aims at recreating a realistic digital application of the instrument through physical modelling synthesis, in the form of digital waveguides

II. Digital Waveguides

Proposed by Smith in [3], digital waveguide methods follow a different path from the ones based on numerical integration of the wave equation. In this implementation, the wave equation is first solved in a general way to obtain travelling waves in the medium. In the lossless case, a travelling wave between two points in the medium can be implemented using nothing but a delay line. The advantage of this is that, operating with Linear Time-Invariant systems (LTI), most of the simulation still consists of delay lines, keeping the computational costs very low. The following explanations are taken from [3], where further explorations could be found.

A. The Ideal Vibrating String

Fully derived in [11] and in most textbooks on acoustic, the wave equation for an ideal (lossless, linear and flexible) string is given by:

$$\frac{\partial^2 y}{\partial t^2} = c^2 \frac{\partial^2 y}{\partial x^2} \tag{1}$$

where $y(t, x)$ represents the vertical string displacement over time t and horizontal space x, and c is the speed of the wave propagating in the string.

Following Bilbao's implementation, the former equation could also be written as:

$$Ky'' = \epsilon \ddot{y} \tag{2}$$

where $K \triangleq$ string tension, $\epsilon \triangleq$ linear mass density, $y \triangleq y(t, x)$ string displacement, $\ddot{y} \triangleq \frac{\partial^2 y}{\partial t^2}$ and $y'' \triangleq \frac{\partial^2 y}{\partial x^2}$. In the following form, the wave equation can be interpreted as a statement of Netwon's second law $\vec{F} = m\vec{a}$ on a microscopic scale. Considering transverse vibration of the string, the force is given by the string tension times the curvature of the string (Ky'') and it is balanced by the mass times the transverse acceleration ($\epsilon\ddot{y}$). For the physical modeling of musical instruments, this is the most common equation and can be applied to any perfect elastic medium which is displaced along one dimension, describing transverse vibration in an ideal string, longitudinal vibration in an ideal bar or pressure in an acoustic tube [7] [12]. For example, the air column of a clarinet or organ pipe can be modeled using the one-dimensional wave equation, substituting air pressure deviation for string displacement, and longitudinal volume velocity of air in the bore for transverse velocity on the string. We refer to this general class of media as *one-dimensional waveguides*, though extensions to more dimensions are mathematically possible [3].

B. Travelling-Wave Solution

The wave equation can be solved by any string shape which travels to the left or right with speed $c = \sqrt{K/\epsilon}$. If we denote right-going travelling waves as $y_r(x-ct)$ and left-going travelling waves $y_l(x+ct)$, then the general solution for the lossless one-dimensional wave equation can be expressed as:

$$y(x, t) = y_r(x - ct) + y_l(x + ct) \tag{3}$$

while an example of the appearance of the travelling wave components is shown in Figure 2.

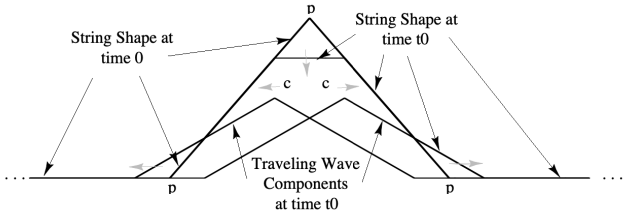

Fig. 2. An example of the appearance of the travelling wave components shortly after plucking an infinitely long string at three points as shown in [3]. The plucking points are labeled by "p" and are plucked simultaneously, producing an initial triangular displacement. This initial displacement is modeled as the sum of two identical triangular pulses which at $t = 0$ are exactly on top of each other but then begin to separate at $t = t0$.

C. Sampling the Travelling Waves

To carry the travelling-wave into the digital domain, it is necessary to sample the travelling-wave amplitudes at intervals of T seconds, corresponding to a sampling rate $f_s = 1/T$ (samples per second). The spatial sampling index X is defined as $X = cT$ (meters), representing the distance sound propagates in one temporal sampling interval T. In air, where c is the speed of sound ($331m/s$), for a $f_s = 44100Hz$, $X = 331/44100 = 7.5mm$ of spatial sampling interval.

Sampling is carried out by the following change of variables:

$$\begin{aligned} x \to x_m &= mX \\ t \to t_n &= nT \end{aligned} \tag{4}$$

Substituting them into equation 3 gives:

$$\begin{aligned} y(t_n, x_m) &= y_r(t_n - x_m/c) + y_l(t_n + x_m/c) \\ &= y_r(nT - mX/c) + y_l(nT + mX/c) \\ &= y_r[(n-m)T] + y_l[(n+m)T] \end{aligned} \tag{5}$$

Since T multiplies all the arguments we can suppress it by defining

$$y^+ \triangleq y_r(nT) \qquad y^- \triangleq y_l(nT) \tag{6}$$

Now, the left- and right-going travelling waves could be summed into:

$$y(t_n, x_m) = y^+(n-m) + y^-(n+m) \tag{7}$$

Any ideal one-dimensional waveguide can be simulated in this way. Here, the term $y_r[(n-m)T] = y^+(n-m)$ can be thought as the output of an m-sample delay line whose input is $y^+(n)$, resulting in the right-going component in the upper rail of Figure 3. Similarly, the term $y_l[(n+m)T] = y^-(n+m)$ can be thought as the input of an m-sample delay line whose output is $y^-(n)$, resulting in the left-going component in the lower rail in the same Figure.

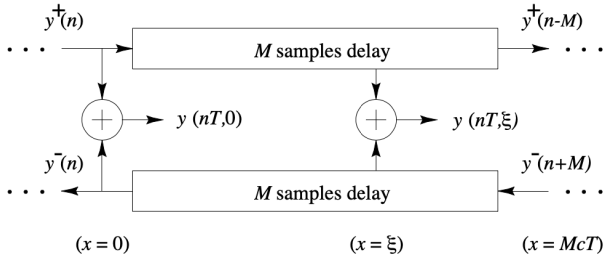

Fig. 3. Simplified picture of ideal digital waveguide as shown in [3]. The figure emphasizes that an ideal and lossless waveguide is simulated by a bidirectional delay line.

D. Rigid Terminations

A rigid termination is the simplest case of a string termination. It imposes the constraint that the string cannot move at all at the termination. If we terminate a length L ideal string at $x = 0$ and $x = L$ we have the so-called *boundary conditions*.

$$y(t,0) = 0 \qquad y(t,L) = 0 \tag{8}$$

Since $y(t,0) = y_r(t) + y_l(t) = y^+(t/T) + y^+(t/T)$ and $y(t,L) = y_r(t - L/c) + y_l(t + L/c)$, the constraints of the sampled travelling waves becomes:

$$\begin{aligned} y^+(n) &= -y^-(n) \\ y^-(n + N/2) &= -y^+(n - N/2) \end{aligned} \tag{9}$$

where $N \triangleq 2L/X$ is the time in samples to propagate from one end of the string to the other and back (total string loop delay).

The ideal plucked string with rigid terminations for the bridge and the nut, implemented in this project, is then defined as a string with initial displacement and zero initial velocity. An example of an initial pluck excitation in this digital waveguide string model is shown in Figure 4

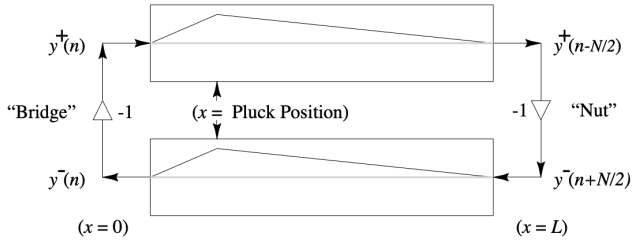

Fig. 4. Initial conditions for the ideal plucked string as shown in [3].

III. Implementation

A. Waveguides in Faust

The Faust Physical Modeling ToolKit is a collection tools to facilitate the design of physical models with the Faust programming language [13]. In order to be modular and to fit the digital waveguide theory proposed in Section II, instrument parts implemented in the Faust `physmodels.lib` must be bidirectional. The block-diagram oriented syntax of Faust [14] allows to easily create chains of blocks going form left to right using the `:` operator and from right to left using $\sim$. However, since $\sim$ creates a feedback signal, it can't really be used to create bi-directional blocks directly [15]. High level elements of the Faust physical modeling library are all based on a series of functions allowing to implement algorithm by using a bi-directional block diagram approach. Each block has 3 inputs and 3 outputs and can be used within a `chain`. The first input and output correspond to left-going waves, the second input and output carry right-going waves, and the third input and output can be used to send a signal to the end of the algorithm. To connect blocks together, the `chain` function must be used. Thus we could have a simple connection of two empty blocks (`simpleBlock`) with a gain block with gain value `g` by simply writing:

```
foo = chain(simpleBlock : gainBlock(g)
: simpleBlock);
```

A visual representation of this is shown in Figure 5, where `chain` essentially reverses the orientation of the second input and output of each block without ending the chain. Thus `foo` here becomes a new bidirectional block that can in turn be used within a new chain. This is the main Faust environment that contains and connects the model elements as blocks.

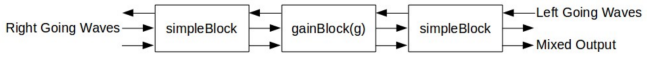

Fig. 5. Graphical representation of the bidirectional chain connector between two empty blocks and a gain block in Faust, as shown in [15].

To implement a digital waveguide in Faust we thus need two delay lines in parallel and an empty channel for a potential output signal. In doing so, we could implement two blocks using 4th order Lagrange interpolation fractional delays. The length of the waveguide is controlled by changing the sample length of the delay lines, derived from the desired frequency:

```
period = (ma.SR/freq/2)-7.5;
p0a = period * position;
p0b = period * (1-position);
```

Since we want to control the tuning of each string, periods for the strings should be defined and sent as arguments to the delays:

```
block0a = de.fdelay4(1024, p0a)
,de.fdelay4(1024, p0a),_;
block0b = de.fdelay4(1024, p0b)
,de.fdelay4(1024, p0b),_;
```

Notice that the initial period, computed as number of samples, has a compensation value of -7.5. This was set tuning the value by ear, since when dealing with exact frequencies, the model implemented using waveguides in Faust is not fine tuned, because each block within a chain will induce a one sample delay in both directions, resulting in a frequency variation.

The desired frequency is set by tuning a slider as a graphical element:

```
freq = hslider("Frequency", 146.83, 20,
1000, 0.01) : si.smoo; // D4
position = 0.9;
gate = button("Play");
```

Signals can be injected anywhere within a chain by using the `in(x)` function, where `x` is a signal to insert between the two string segments (the two blocks previously defined). Typically, the length of one string segment could be changed in function of the other to control the position of excitation on the virtual string. The function `out` works the same way as `in` and bridges the signal of left-going and right-going waves at a specific location in a chain to the output channel. It can be viewed as a pickup on an electric guitar string whose location can be controlled independently from the excitation position or, generally, the position where we listen to the sound of a certain string. The model, using the `chain` function for a single string will look like this:

```
model(x) = endChain(
    chain(
        lTermination(*(-0.995) :
        fi.lowpass(4, 10000),
        basicBlock) :
        block0a :
        in(x) :
        out :
        block0b :
        rTermination(basicBlock, *(-0.995)
        : fi.lowpass(4, 10000))
    )
);
```

Here, `endChain` is used to terminate a chain. Furthermore, since any model needs to resonate, we need to connect right-going waves to left-going waves and vice versa. This can be easily done using the `lTermination` and the `rTermination` functions, as shown in Figure 6, where `b` is a bidirectional block (it could be a chain, of course) and `a` is function with one input and one output.

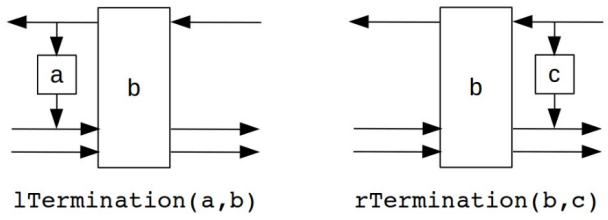

Fig. 6. Graphical representation of the left and right rigid terminations in Faust, as shown in [15].

To implement ideal lossless rigid string terminations, as previously described in Section II-D, each termination takes as an input a basic empty block and a multiplication by -0.995, as previously shown in code. This value implements a lossless reflexion with a phase inversion, satisfying the boundary conditions. Additionally, a damping factor has been implemented using a low pass filter.

The full model, resulting in 13 iterations of the waveguide just described, each one with the corresponding frequency for the right tuning of each string, is shown in Figure 7.

B. From Faust to Max/MSP

After being implemented in Faust, the model was compiled for Max/MSP [16] using the embedded `faust2max` tool compiler. This feature generates a native standalone application, namely a `mxo` Max object, a Max patcher and a `js` object that implements the GUI parameters previously described in

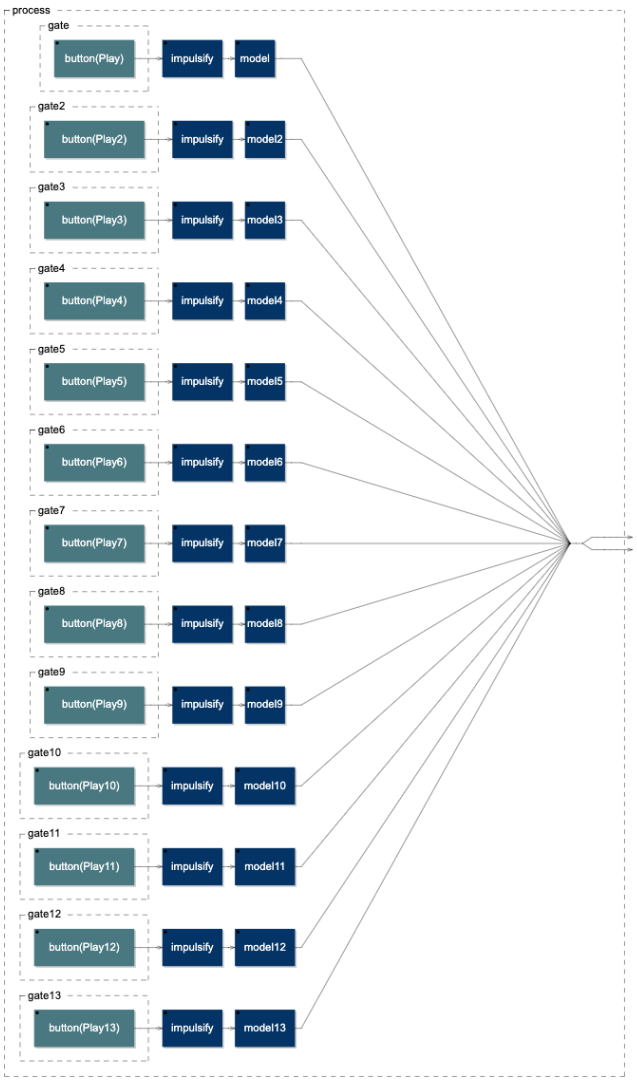

Fig. 7. Block diagram of the full model of the Koto implemented in Faust as the iterations of 13 digital waveguide models, one for each string of the instrument.

the Faust code (i.e. faders, buttons, dials). The motivation for this has been given by the decision to control the model using OpenSoundControl (OSC) [17], a data transport specification (an encoding) for realtime message communication among applications and hardware. OSC can be understood as a more flexible alternative to MIDI, as it clears away many of the ideological and hardware constraints inherent to MIDI in favor of a open-ended, user-defined address-space model that provides arbitrary parametric control via standard networking hardware [18] [19].

C. OSC Data handling

To send OSC data from a mobile device to a computer running Max/MSP, TouchOSC was used. TouchOSC [20] is a modular control surface toolkit for designing and constructing custom controllers that can be used on a multitude of operating systems and devices. TouchOSC can be used on touch-screen mobile devices as well as desktop operating systems using traditional input methods and can communicate with other software and hardware using the MIDI and OpenSoundControl protocols in a variety of ways and via many different types of wired and wireless connections simultaneously. Furthermore, a free editor is available to create and customize own layouts for TouchOSC, like the one used in this project, as shown in Figure 8. The layout, consisting of two columns of aligned long buttons, aims at recreating a virtual visual representation of the koto, from the player's perspective. In this way, the right part of the interface represents the strings to pluck, while the left part is dedicated to the bending of each string. When a button is pressed, an OSC message is sent to a dedicate Max/MSP patch, taking care of the routing of the message through UDP network. The requirement for this procedure to succeed is that both the device running TouchOSC (in this case an iPad) and the PC must be connected to the same WiFi network, and the device must know the IPv4 address of the PC. After matching the destination port of the device with the source port of the PC, Max/MSP can receive the data packets through the object `udpreceive`, with the number of the selected port as an argument, as shown in Figures 9 and 11.

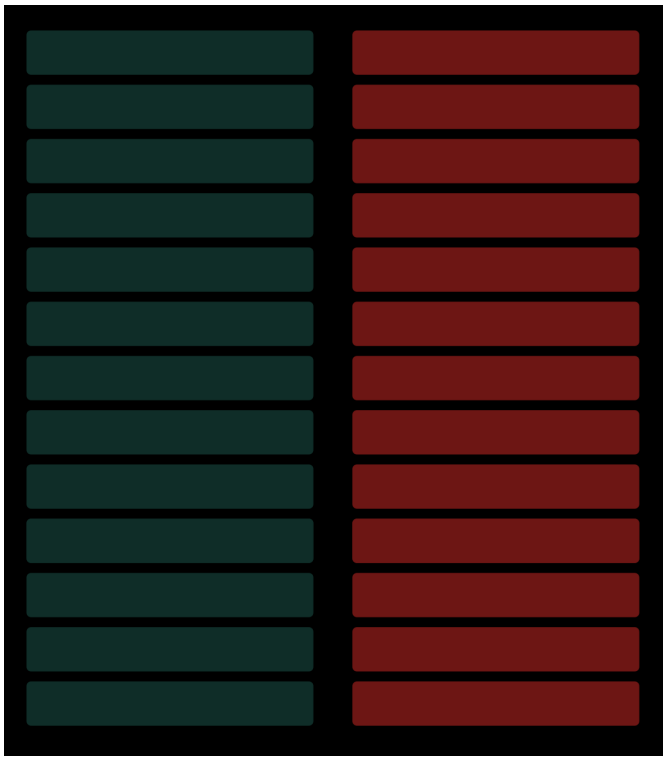

Fig. 8. Interface for the application, designed with the TouchOSC editor. On the right side (red) the strings can be plucked, individually or simultaneously. Pressing on the left side (blue) activates the bending of the corresponding strings.

D. Max/MSP Patches

In this section I will give a brief explanation of the four Max/MSP subpatches consisting of the handling and routing of OSC messages to the synthesis parameter. A full overview of the main patch is presented in Figure 13.

1) routePlay: as shown in Figure 9, this patch simply reads through the messages received via `udpreceive` on port 8000 and routes the messages related to the `play` of each string. The `route` command selects a certain output based on input matching and outputs a bang when this is true.

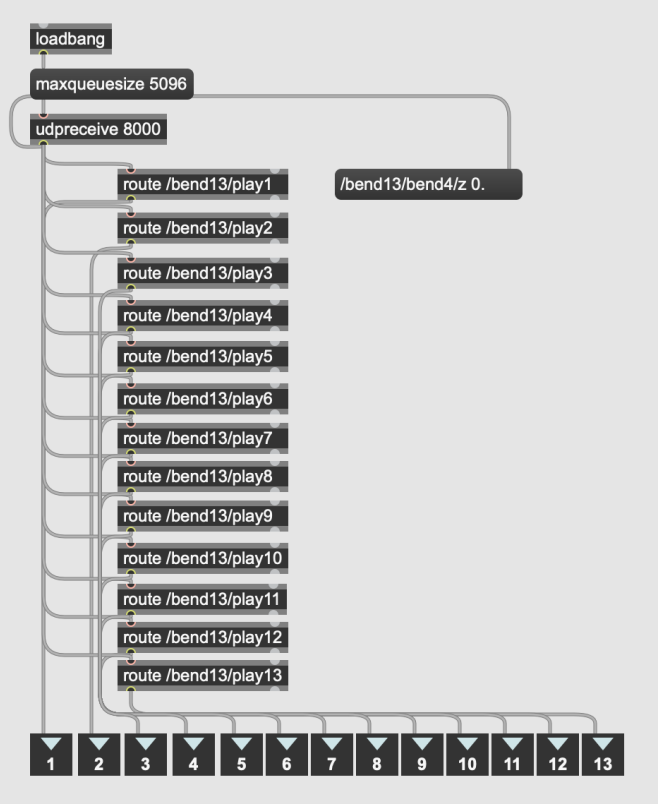

Fig. 9. Max/MSP subpatch in charge of the routing of `play` OSC messages from the iPad. This works as a sorting layer between the touch interface and the synthesis engine.

2) playStrings: this patch receives as inlets the routed bangs computed from the *routePlay* patch. As presented in Figure 10, every bang is sent to a toggle assigned to each individual string. When the bang for a certain string is received, the toggle sends the relative `play` message to the main `mxo` object that has been generated from the `faust2max` conversion, which represents the synthesis engine.

3) routeBend: similarly to *routePlay*, this patch routes the OSC messages received through `udpreceive` on port 8000 to check if the buttons for the bending of the strings have been pressed. As shown in Figure 11, if this is true for a particular string, a ramp consisting of the starting frequency and the final

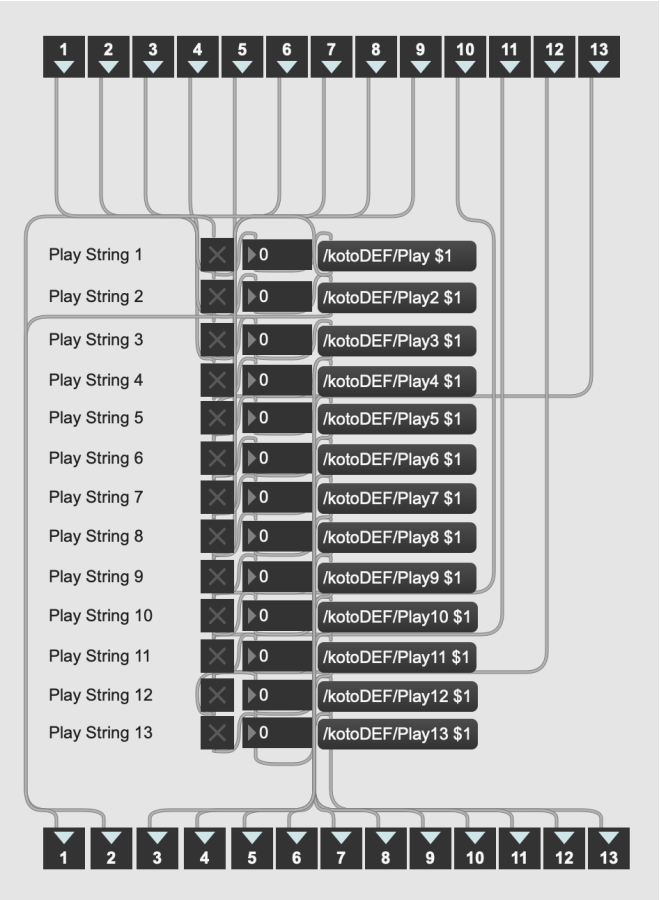

Fig. 10. Max/MSP subpatch for string playing. The messages routed from *routePlay* are sent to the relative strings and then to the physical modelling synthesis engine.

frequency of the bending is generated over a range of $100ms$ and sent through the relative outlet of the patch.

4) bendStrings: this patch takes as an input the messages generated from *routeBend* in the form of ramps of varying frequencies. Those are visualized through a slider and sent to the main `mxo` object using the generated `frequency` message relative to the bent strings (see Figure 12).

IV. Results

The main Max/MSP patch containing the four described subpatches and the `mxo` object generated from `faust2max` is presented in Figure 13. A reverb has been added to enhance the sound qualities of the instruments and two different sets of traditional tunings of the strings are available.

A video demo of the application, in the context of playing a traditional japanese song, is available[1].

[1] https://drive.google.com/file/d/1oY_NUpJavctM4Si-lAakjxL7B-LZsmKF/view?usp=sharing

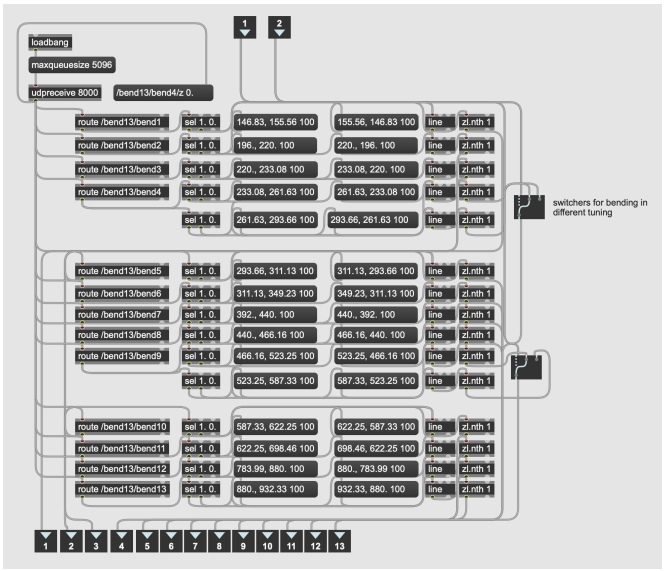

Fig. 11. Max/MSP subpatch in charge of routing the bending messages from TouchOSC. Frequency ramps are specifically designed for each string, relatively to the amount of bending.

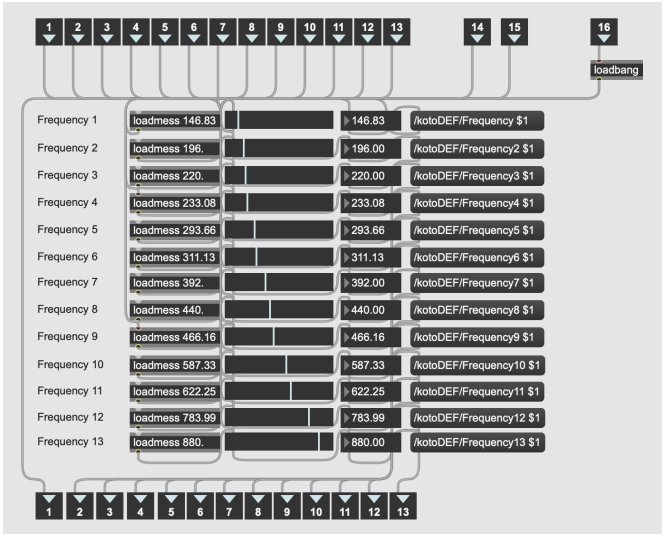

Fig. 12. Max/MSP subpatch receiving the frequency ramps generated by *routeBend* and sending them to the relative bent strings.

V. Discussion

The goal of this project was to design a digital version of the Koto, an ancient japanese instrument, using physical modelling synthesis techniques, specifically digitally waveguides. The result is a Max/MSP application that can be played through a mobile device (in this case an iPad) using a custom TouchOSC interface layout.

The advantage of this architecture is that the interface and the sound engine are kept separated, resulting in a optimal application for the museum itself. In this case, an iPad with the interface layout could be placed in front the of the glass containing the original instrument, free for the visitors to play it. The visual representation of the interface, with the plucking part of the string on the right and the bending side on the left, spontaneously represents the real instrument, especially if the interface is placed just in front of it, aiding the visual linking and representation of the Koto. The only requisite for this architecture to work is to have both the devices connected to the same WiFi network, another optimal criterion for the museum to host it.

Nevertheless, future works could enhance the bending possibilities, likely with the use of a Force-Sensing Resistor (FSR) to control different amounts of bending. Further explorations need to be accounted also for the affordance of the TouchOSC interface in the case of fast playing, where drops of udp packets occur, resulting sometimes in latency of the produced sound related to the pressed buttons.

VI. Conclusions

The present implementation of physical modelling techniques contributes to a larger context of studies on digital reproduction of existing real world musical instruments. It is hoped that the research done during this study can work as a personal foundation for future investigations in these fields, as well as in the area of tools for digital assisted composition and improvisation.

References

[1] V. Valimaki and T. Takala, "Virtual musical instruments - natural sound using physical models," *Organised Sound*, vol. 1, p. 75, 1996.

[2] G. D. Poli and D. Rocchesso, "Physically based sound modelling," *Organised Sound*, vol. 3, pp. 61–76, 4 1998.

[3] J. O. Smith, "Physical modeling using digital waveguides," *Computer Music Journal*, vol. 16, p. 74, 24 1992.

[4] C. Cadoz, A. Luciani, and J. L. Florens, "Cordis-anima. a modeling and simulation system for sound and image synthesis. the general formalism," *Computer Music Journal*, vol. 17, pp. 19–29, 1993.

[5] P. R. Cook, "Real sound synthesis for interactive applications."

[6] J. M. Adrien, "The missing link: modal synthesis | representations of musical signals." [Online]. Available: https://dl.acm.org/doi/10.5555/131150.131158

[7] S. D. Bilbao, *Numerical sound synthesis : finite difference schemes and simulation in musical acoustics*. Wiley Publishing, 2009.

[8] S. Bilbao, C. Desvages, M. Ducceschi, B. Hamilton, R. Harrison-Harsley, A. Torin, and C. Webb, "Physical modeling, algorithms, and sound synthesis: The ness project," vol. 43, pp. 15–30, 2019. [Online]. Available: www.ness-music.eu.

[9] M. V. Mathews, "The digital computer as a musical instrument," *Science*, vol. 142, pp. 553–557, 1963.

[10] "The danish music museum." [Online]. Available: https://en.natmus.dk/museums-and-palaces/the-danish-music-museum/

[11] P. M. Morse, "Vibration and sound," 1936.

[12] S. Willemsen, *The Emulated Ensemble, Real-Time Simulation of Musical Instruments using Finite-Difference Time-Domain Methods*. Aalborg Universitet, 2021.

[13] R. Michon, "Romain michon - faust physical modeling toolkit." [Online]. Available: https://ccrma.stanford.edu/~rmichon/pmFaust/

[14] Y. Orlarey, D. Fober, and S. Letz, "Faust : an efficient functional approach to dsp programming - archive ouverte hal," *New Computational Paradigms for Computer Music*, pp. 65–96, 2009.

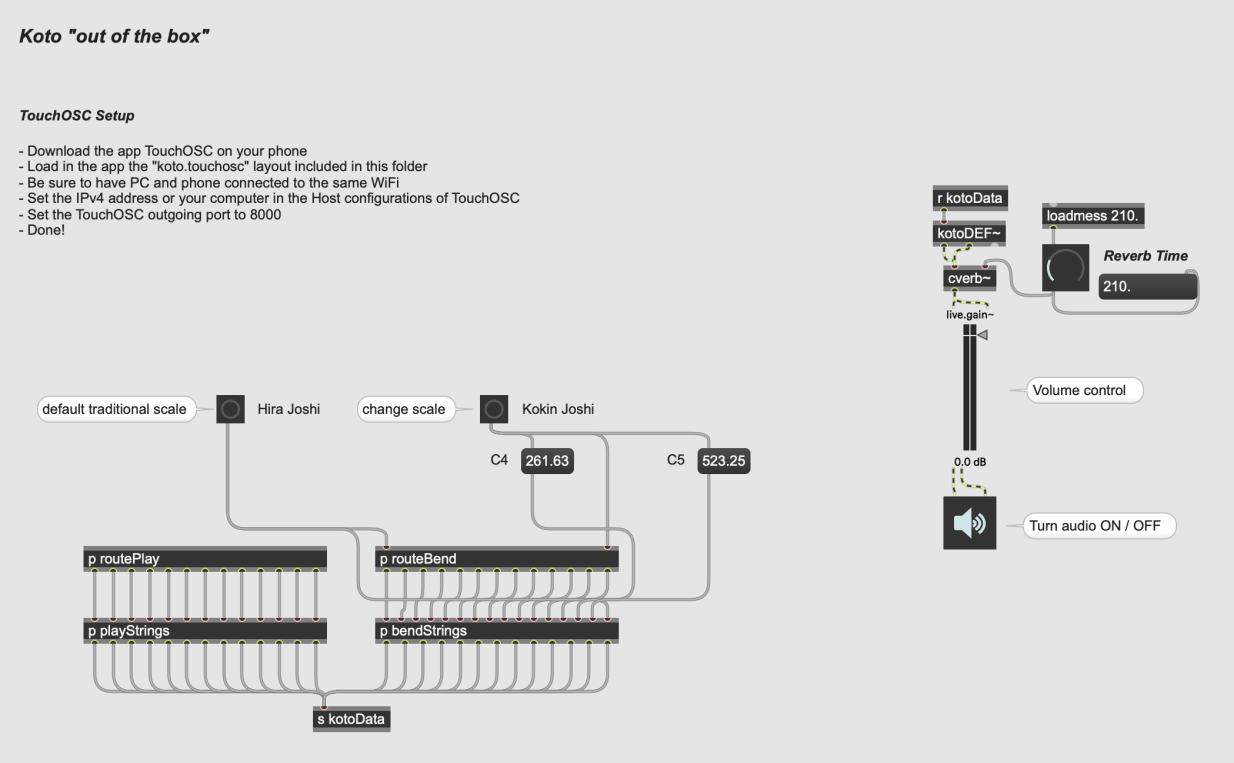

Fig. 13. Max/MSP main patch, containing the `mxo` object in charge of the physical modelling synthesis, as well as the four subpatches for routing of OSC messages. A reverb has been added and two different traditional tunings could be selected through dedicated buttons.

[15] R. Michon, "Making physical models of musical instruments with faust." [Online]. Available: https://ccrma.stanford.edu/~rmichon/faustTutorials/#making-physical-models-of-musical-instruments-with-faust

[16] "Max/MSP, Cycling '74." [Online]. Available: https://cycling74.com/

[17] "OpenSoundControl." [Online]. Available: https://ccrma.stanford.edu/groups/osc/index.html

[18] M. Wright and A. Freed, "Open soundcontrol: A new protocol for communicating with sound synthesizers." [Online]. Available: http://www.cnmat.berkeley.edu/People

[19] A. Freed and A. Schmeder, "Features and future of open sound control version 1.1 for nime."

[20] "TouchOSC, Fully modular touch control surface for OSC and MIDI." [Online]. Available: https://hexler.net/touchosc-mk1