KEYTAR: MELODIC CONTROL OF MULTISENSORY FEEDBACK FROM VIRTUAL STRINGS

Federico Fontana [*]
HCI Lab
University of Udine
Udine, Italy
federico.fontana@uniud.it

Andrea Passalenti [†]
HCI Lab
University of Udine
Udine, Italy
passalenti.andrea@spes.uniud.it

Stefania Serafin [‡]
Multisensory Experience Lab
Aalborg University Copenhagen
Copenhagen, Denmark
sts@create.aau.dk

Razvan Paisa [§]
Multisensory Experience Lab
Aalborg University Copenhagen
Copenhagen, Denmark
rpa@create.aau.dk

ABSTRACT

A multisensory virtual environment has been designed, aiming at recreating a realistic interaction with a set of vibrating strings. Haptic, auditory and visual cues progressively istantiate the environment: force and tactile feedback are provided by a robotic arm reporting for string reaction, string surface properties, and furthermore defining the physical touchpoint in form of a virtual plectrum embodied by the arm stylus. Auditory feedback is instantaneously synthesized as a result of the contacts of this plectrum against the strings, reproducing guitar sounds. A simple visual scenario contextualizes the plectrum in action along with the vibrating strings. Notes and chords are selected using a keyboard controller, in ways that one hand is engaged in the creation of a melody while the other hand plucks virtual strings. Such components have been integrated within the Unity3D simulation environment for game development, and run altogether on a PC. As also declared by a group of users testing a monophonic Keytar prototype with no keyboard control, the most significant contribution to the realism of the strings is given by the haptic feedback, in particular by the textural nuances that the robotic arm synthesizes while reproducing physical attributes of a metal surface. Their opinion, hence, argues in favor of the importance of factors others than auditory feedback for the design of new musical interfaces.

1. INTRODUCTION

Along with the continuous miniaturization and proliferation of digital hardware, the research domain of computer interfaces for musical performance is increasingly taking advantage of the growing market of haptic devices. Traditionally confined to the loudspeaker set, the output channel in these interfaces now often incorporates force and tactile feedback targeting the performer [?] and/or the audience [?]. In spite of the debate around the effects that synthetic tactile cues have on the precision of the execution [?] and in general on the quality of the performance [?], no doubt exists on the impact that haptic feedback has as a conveyor of realism, engagement and acceptability of the interface [?]. This design trend has developed to the point that a specific research line has been recently coined: musical haptics [?].

In parallel to such hardware opportunities, several software architectures have become available. These architectures contain libraries which provide relatively fast access to functionalities that, until few years ago, needed significant labor of researchers interested in programming a new musical interface. Conversely, modern software development environments such as Max/MSP or Unity3D have enough resources in-house for realizing digital instrument prototypes yielding a credible image of the final application. The latter software, in particular, already in its freeware version offers an unprecedented support to existing multimedia devices in the context of a visual or, at the developer's convenience, traditional programming environment. Originally intended for computer game design, Unity3D is being embraced by an increasing community of programmers with interests in virtual and augmented reality, as well as cultural and artistic applications on multimodal technologies.

Among the numerous *assets* Unity3D makes available, basic objects can be found for defining elastic strings which dynamically respond to colliding bodies and, at that moment, play an audio sample or launch a sound synthesis algorithm. Such a core scenario can receive messages by external devices including Musical Instrument Digital Interface (MIDI)-based controllers like a keyboard, and haptic devices like e.g. a robotic arm.

Using these components we have started a project, now called Keytar, aiming at understanding the perceptual importance of haptic, auditory and visual feedback during a point-wise interaction with a virtual string set. This idea is not new: multisensory environments with visual, auditory and haptic feedback have for example been developed for several decades in the context of the Cordis-Anima framework [?]. Our approach, however, aims at an efficient yet accurate simulation in a widely accessible virtual reality environment. We have reproduced a standard and a bass guitar employing different string sets, and then compared sample-based rather than synthesized sounds under different visual viewpoints

[*] Concept, writing, lab facilities
[†] Design, development, implementation, writing
[‡] Writing, funding dissemination
[§] Implementation

of the instruments. Later we have integrated melodic control via MIDI keyboard. This paper reports the current status of the Keytar project, documenting what has been done and what is ongoing based on the lessons learnt so far: Sec. 2 specifies the hardware in use; Sec. 3 describes the software component realizing communication with the devices, haptic workspace, sound synthesis and visualization; Sec. 4 explains how Keytar was first calibrated and then preliminary tested; Sec. 5 concludes the paper.

2. PHYSICAL INTERFACE

The Keytar interface is shown in Fig. 1. While users pluck a string through the robotic arm using their dominant hand, they simultaneously feel its resistance and textural properties. This way, the plucking force results from a natural perception-and-action process involving users and the string. In the meantime, with the other hand they select notes and/or chords through the keyboard controller. The setup is completed by a screen displaying the action of the virtual plectrum against the vibrating strings, as well as by the audio.

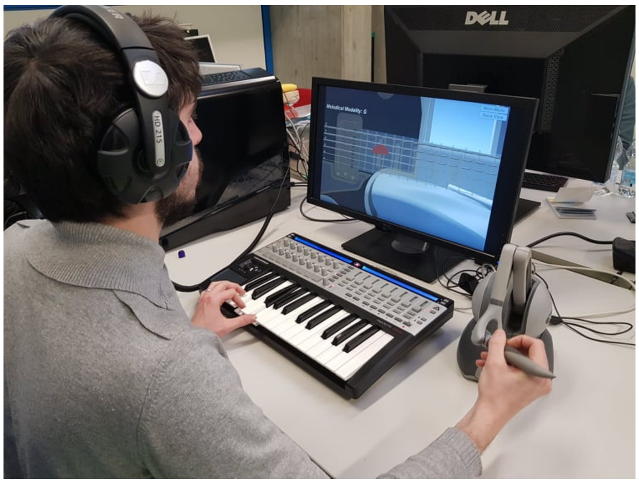


Figure 1: *Keytar: desktop configuration.*

Besides the presence of a monitor and a loudspeaker or headphone set, the most interesting physical component of the output interface is certainly the robotic arm (Fig. 2). A SensAble (now 3D Systems) Phantom Omni was available for this project. The limited force its motors exert on the mechanic arm is compensated by extremely silent and low-latency reactions, making this device still an ideal candidate for virtualizing point-wise manipulations that do not need to render stiff materials. Particularly in the case of elastic string simulation, the Phantom Omni revealed ideal characteristics: since only rotation and lift are actuated, the arm renders elastic reactions while leaving users free to select the angular position of the plectrum at their own taste through the stylus. Even more surprisingly, while rendering the textural properties of wound metal strings its motors produced sounds that are very similar to the scraping effect a plectrum creates when it is slid over a bass guitar string. This acoustic by-product coming from the robotic arm added much realism to the multi-sensory scenario, completing the audio-tactile feedback with nuances whose acoustic component would be otherwise particularly difficult to synthesize and keep synchronized with the haptic channel.

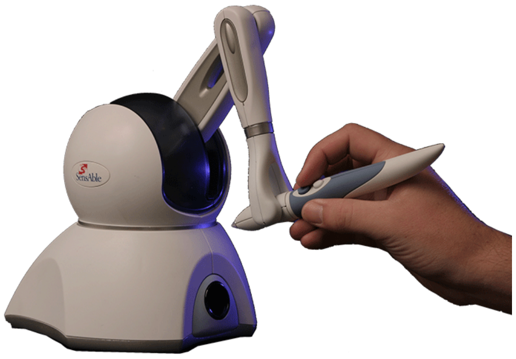


Figure 2: *Phantom Omni robotic arm.*

Concerning the input interface, the robotic arm is once more interesting for its ability to instantaneously detect the stylus position and force exerted on it when its motors are resisting to the arm movement. Position was used to detect when the plectrum collided against the strings. Aside of the robotic arm, a Novation ReMOTE 25SL controller was set to acquire standard MIDI notes across its two-octave keyboard. Finally, two buttons located on the arm stylus at immediate reach of the index finger (both visible in Fig. 2) allowed for selecting major (blue button pressed) or minor (grey button pressed) chords instead of single notes through the keyboard. In other words, users could play chords or alternatively solos respectively by pressing either button or by leaving both buttons depressed. Since there is no obvious mapping from piano to guitar keys, rules were adopted to create a link between the two note sets which will be discussed at the end of Sec. 3.2.

3. SOFTWARE ARCHITECTURE

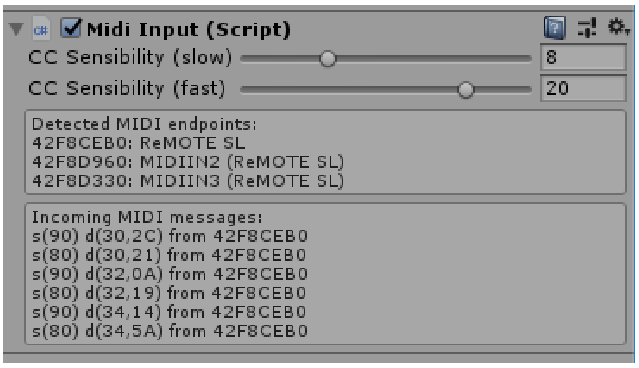
Due to the efficiency of the applications built for many computer architectures, Unity3D is gaining increasing popularity also among musical interface designers. Moreover, this programming environment adds versatility to the development of a software project thanks to its support to a number of peripherals for virtual reality. Finally, software development is made easier by a powerful graphic interface, making it possible to visually program several C# methods and classes that would otherwise need to be traditionally coded through textual input of the instructions. The same interface allows also to preview the application at the point where the project development is.

For the purposes of the Keytar project, Unity3D supported i) communication with the robotic arm and MIDI controller, ii) agile virtual realization of vibrating strings, iii) implementation of collision detection between such strings and a virtual plectrum, iv) access to an algorithm capable of interactively synthesizing string tones, and finally v) relatively easy development of the graphical interface. In the following we browse such components, emphasizing aspects whose knowledge may help researchers in computer music interfaces reproduce the prototype, or take inspiration from Keytar while designing their own instrument.

3.1. Communication with peripherals

Bi-directional communication with the Phantom Omni is made possible by Unity3D's Haptic Plugin for Geomagic OpenHaptics, developed for Windows and (in beta version) for Linux by the Glasgow School of Art's Digital Design Studio. Through this asset it was possible to get the position of the Phantom's arm and simultaneously send control messages to its motors, hence defining the instantaneous behavior of the robotic device. In particular, Haptic Plugin manages force through higher level parameters: *stiffness*, *damping*, *static friction*, *dynamic friction*, *tangential stiffness*, *tangential damping*, *punctured static friction*, *punctured dynamic friction*, and *mass*. Expansions of this palette appearing in the newer versions of Unity3D have outmatched such parameters, in practice forbidding the Phantom Omni to interact with dynamic (i.e., moving) virtual objects.

Concerning the communication with the keyboard, Keijiro Takahashi's MIDI Jack is an open project[1] enabling Unity3D to acquire and interpret some types of MIDI events coming from a controller that sends messages in this protocol. In particular, it features automatic recognition of input MIDI devices and returns non-zero velocity values of notes when a key is pressed. This software module immediately establishes a communication between the controller and the note selector running in Keytar (see Fig. 3). Though,



```
Midi Input (Script)
CC Sensibility (slow)    8
CC Sensibility (fast)    20

Detected MIDI endpoints:
42F8CEB0: ReMOTE SL
42F8D960: MIDIIN2 (ReMOTE SL)
42F8D330: MIDIIN3 (ReMOTE SL)

Incoming MIDI messages:
s(90) d(30,2C) from 42F8CEB0
s(80) d(30,21) from 42F8CEB0
s(90) d(32,0A) from 42F8CEB0
s(80) d(32,19) from 42F8CEB0
s(90) d(34,14) from 42F8CEB0
s(80) d(34,5A) from 42F8CEB0
```

Figure 3: *MIDI input inspector in Unity3D.*

if the application is ported on another PC then care must be taken in keeping the MIDI channel that was chosen during building. Future versions of Keytar will accept this value as an initial argument.

3.2. Strings, plectrum, and collision detection

Each string was created using *Cylinder* objects. Such objects are instantiated by a primitive class of Unity3D once their length, diameter and position in the virtual space are set. The plectrum was instead obtained by putting one passive and one *active* object together: the former gave visual appearance by realizing the traditional shape, while the latter was responsible of the collision against the strings and was instantiated as a *Sphere* object having minimum (i.e. almost point-wise) diameter.

The little sphere was placed on one edge of the plectrum and then made invisible by disabling its display. Like we did also for each string, the sphere activity was enabled by interfacing it with the classes *RigidBody* and *Collider*, containing methods for the real-time detection and management of collisions. Since users often move the plectrum beyond the plane containing the strings, the interaction was made more robust by positioning an active surface just beyond this plane. This object, invisible to the user and having the same role as a fretboard, cannot be crossed by the sphere and hence avoids occasional backward collisions entangling the plectrum behind the strings.

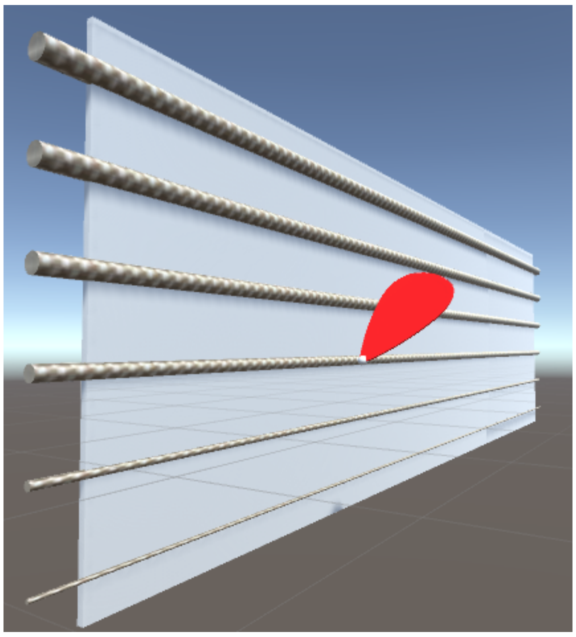


Figure 4: *Haptic workspace in Keytar. The colliding sphere is visible as a white tip on the edge of the plectrum.*

Active objects become part of the haptic workspace, accessible to the robotic arm, if they are labeled as *Touchable*. In this way, their position and shape define areas of the virtual scenario that become inaccessible to the tip of the arm stylus, as obviously no more than one solid object at a time can occupy a single volume of the workspace. In parallel such object surfaces inherit the haptic parameters listed in Sec. 3.1, which are used at runtime while computing the collisions occurring between touchable objects. Fig. 4 shows the final appearance of the haptic workspace. Initially we had labeled the entirely visible plectrum as touchable, i.e. with no distinction between active and passive region. Unfortunately, distributing contact areas across objects having irregular shapes can encumber the computation of the haptic workspace with occasional crashes of the application especially when running on slower machines. This issue was solved in Keytar by tying a tiny touchable sphere to the plectrum.

Clearly, this simplification leads to visuo-haptic mismatching as soon as a passive region of the plectrum intersects with the string. The net effect was that the plectrum often went through strings with no apparent contact. A more subtle visuo-haptic mismatch manifests when the active edge of the plectrum touches, but does not penetrate a string enough: in this case the active objects interact each other as one could clearly see also from the screen, but no collision event is triggered inside the haptic workspace. The latter problem was solved by wrapping each string with an invisible, inactive meanwhile touchable shield, which was set to be as thick as the diameter of the sphere active in the plectrum—see Fig. 5. This shield in fact paired the contact point positions set by the *Touchable* and *Collider* methods. Such string wrappers also attenuate the former problem, as their inclusion made it more difficult to dip the plectrum into the string without production of reactive force from the robotic arm due to collision of the sphere

[1]https://github.com/keijiro/MidiJack

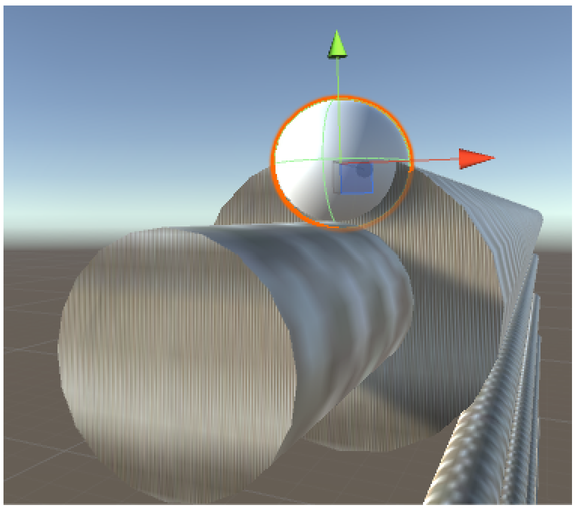


Figure 5: *Mismatching positions of visual and haptic collision point. Solution using string wrappers.*

against the wrapper.

A strategy for estimating each string velocity after a collision was needed. To this regard, Haptic Plugin gives runtime access to the contact point coordinates immediately before (b) and after (a) every collision. We decided to estimate velocity as the distance between such two positions divided by the haptic frame rate, readable from the static property *Time.deltaTime*:

$$\text{velocity} = \frac{\sqrt{(x_a - x_b)^2 + (y_a - y_b)^2 + (z_a - z_b)^2}}{Time.deltaTime}.$$

Velocity was then used to determine the amplitude of vibration in each string.

3.3. Sample-based, interactive, and physical sounds

In Unity3D, sound source and listening point are respectively defined by *AudioSource* and *AudioListener* objects. In the case of a guitar simulation the Keytar workspace contained six sound sources, one for each string. In parallel, a default audio listener was associated to the standard *Camera* object, corresponding to the viewpoint in the virtual scenario. On their way from the sources to the listening point, sounds can be additionally routed in an *AudioMixer* object. On top of mixing down such sources to pick-up points for the audio card, the *AudioMixer* class allows for interconnection of, and interaction at runtime with several sound processing methods including filters, reverberators, compressors, and further digital audio effects. Alone, these methods turn Unity3D in an effective digital audio workstation.

We kept the *AudioMixer* object to the simplest, by just picking up sounds from the strings and mixing them down before reproduction from the virtual listening point. The corresponding console, shown in Fig. 6, was sufficient for reproducing a bank of guitar samples we use in the initial prototype. This prototype had no note selection, and for this reason just six tones corresponding to the freely vibrating strings were recorded to form this bank, from a guitar playing at several dynamic levels.

Later, when the note selection was introduced, we switched to interactive synthesis by including the well-known Karplus-Strong

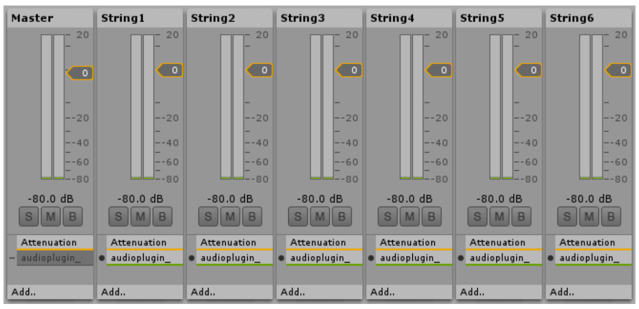
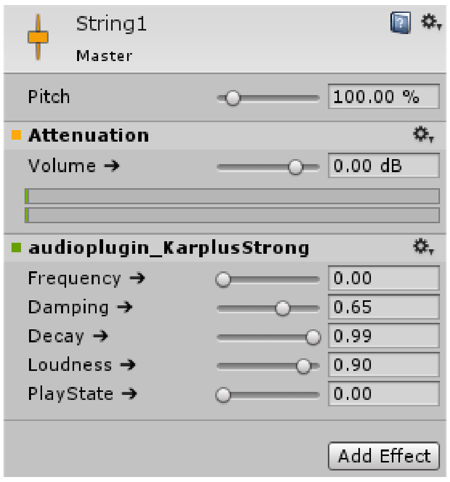


Figure 6: *AudioMixer object in Keytar.*

algorithm[2] in the software architecture [?]. Once imported, this



String1
Master
Pitch 100.00 %
Attenuation
Volume → 0.00 dB
audioplugin_KarplusStrong
Frequency → 0.00
Damping → 0.65
Decay → 0.99
Loudness → 0.90
PlayState → 0.00
Add Effect

Figure 7: *Karplus-Strong control panel in Unity3D.*

synthesizer exposes the control panel as in Fig. 7, containing five parameters. Except for damping, all such parameters can be set at runtime to values between zero and one. For this reason it was necessary to program a method mapping every tone fundamental frequency into this interval. Apart from the freedom of choice of the notes and their timbre using Unity3D's Karplus-Strong—see in particular the simultaneous presence of *damping* and *decay*, alternatively absent in the traditional algorithm instead—the net result of the switch from sample-based to interactive synthesis was a definitely more reactive instrument with empoverished sound quality. Some preliminary sound engineering using standard guitar effects available in Unity3D, however, suggests that it should not be difficult to synthesize acceptable virtual electric guitar sounds with Keytar.

As anticipated in Sec. 2, Keytar features three different playing modes that can be selected through the stylus buttons: solo, harmonic major and harmonic minor. Any switch of these buttons calls a script that activates the corresponding playing mode. In solo mode, the related script simply runs an instance of the Karplus-Strong algorithm at the time when a collision happens, by tuning the synthesizer to the tone that has been set by the controller key—not below the lowest note the excited string can produce, however. The harmonic modes instead launch several instances of the same

[2]`https://github.com/mrmikejones/KarplusStrong`

algorithm, by building major and minor chords based on the last key pression on the controller. This event dictates the fundamental note of corresponding chord. All chords were standard forms based on the chord voicing music theory. They were formed only by tones belonging to the fundamental tonality.

Last, but not least, the acoustic feedback includes real sounds that are a by-product of the effort the robotic arm makes while reproducing the plectrum-string contact. As it can be seen from Fig. 3, each string results by alternating cylinders of two different diameters. The resulting surface geometry hence reproduces a texture assimilable to that of a wound string coating, as it contains regular discontinuities. While following the longitudinal motion of a plectrum scraping along such a string, the motors simulate a motion similar to friction, producing not only haptic, but also acoustic cues of convincing quality, with substantial improvement of the overall realism of the interaction.

3.4. Visual display

In a scenario populated by elements belonging to a typical guitar playing set, the plectrum and the strings are the only animated objects visible from the standard *camera* viewpoint. Plectrum movements directly reproduce the shifts and rotations of the arm stylus, proportionally to the workspace size. Strings visually vibrate thanks to the *Animator* interface, whose methods implement Unity 3D's Mecanim Animation System. This system concatenates basic *Animation* objects along time, together realizing the visual flow visible from the *Camera* object.

In Keytar each *animation* object models an oscillatory transversal shift having specific frequency and decaying amplitude along time. By concatenating few such objects, each string vibrates up and down with decreasing amplitude. This model demands low computational effort in ways that each string position can be refreshed every 10 milliseconds, that is, about 1.6 times the refresh rate of the video. This ratio creates an effect of occasional semi-transparency of the vibrating string, which is beneficial for the realism—see Fig. 8.

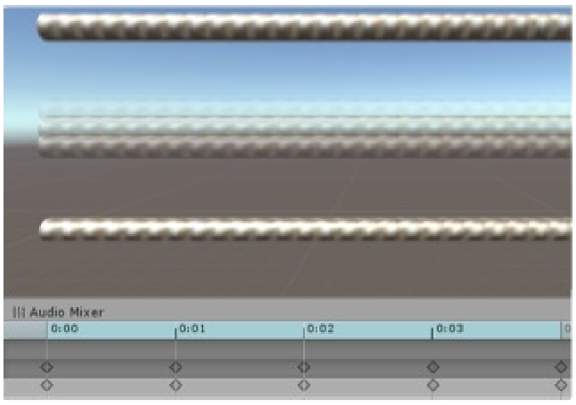


Figure 8: *Semitransparency of one animated string in Keytar.*

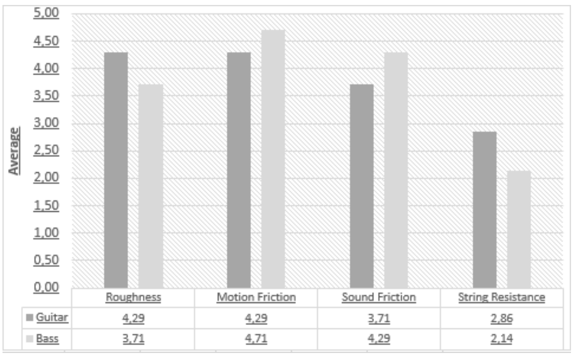
In practice, each collision triggers a script that determines the initial vibration amplitude. Then, vibrations are rendered by an *Animator* method that destroys a visual string while cloning it into a shifted instance. This workflow continues until the original string returns at rest or, conversely, a new collision happens against the initial string position. In other words, the clones were not interfaced to the *Collider* class nor did they inherit any *Touchable* property, as the definition of a haptic workspace evolving dynamically at such a fast refresh rate would suffer from the instability mentioned in 3.1. This limit defers any realization of vibratory feedback in Keytar to a version capable of driving vibrotactile devices not using data about the interaction point, but rather higher level information about amplitude and pitch of the string vibrating in contact with the plectrum.

In spite of its efficiency and versatile management of collision events, an accurate visual rendering of the string would require to realize a rectangular surface that shrinks, and becomes progressively less semitrasparent across time until reducing to a solid string at rest position. Finally, the virtual string did not model rigid edges. This approximation results in strings moving up and down as stiff bars would do, with apparent artefacts whose removal will eventually require a different design of the visual string.

4. CALIBRATION AND PRELIMINARY TESTING

The strings' physical parameters were tuned by two students, who regularly perform with their guitar and bass in a pop band. Then, Keytar was preliminary tested during two experiments that have been documented in previous papers [**?**, **?**] when Keytar had no polyphonic keyboard control yet. Both such experiments focused on the realism of the multi-sensory interaction with the strings. Here we report limitedly to the answers participants gave concerning the audio-haptic interaction.

In experiment 1, seven participants with different music experience and knowledge of the technology were asked first to pluck a guitar and a bass guitar using a plectrum. Then, they tested Keytar and answered a questionnaire. Regarding the realism of audio-haptic feedback, participants were asked to rate *roughness* of each string surface, *longitudinal motion friction* while sliding the plectrum, *sound friction* during the same sliding, and finally *string resistance*. While the first three attributes were essentially tactile and depended on the vibratory activity of the arm motors, the fourth attribute was kinaesthetic and linked to the force feedback generated by the robotic arm.



	Roughness	Motion Friction	Sound Friction	String Resistance
▨ Guitar	4,29	4,29	3,71	2,86
▨ Bass	3,71	4,71	4,29	2,14

Figure 9: *Experiment 1: Results from haptic evaluation.*

Fig. 9 shows good scores for the first three attributes. Conversely, *string resistance* scored lower. The reason for this dfference might be due to the need to grasp a stylus instead of a plectrum during playing.

Experiment 2 aimed at evaluating the relative importance of force and vibrotactile feedback for the creation of a realistic experience [**?**]. For this reason, the arm stylus was modified for the occasion by applying a 3D printed plectrum on its tip. This physical plectrum was embedded with a Haptuator Mark II vibrotactile actuator by Tactile Labs. Each time a collision occurred, the vibrotactile actuator rendered an impact generated using the Sound Design Toolkit for Max/MSP [**?**]. Twenty-nine subjects were first asked to pluck a real string, then they were exposed to four conditions: no haptic feedback (N), vibrotactile feedback (V, made with the Haptuator), force feedback (F, made with the Phantom Omni), and combination of force and vibrotactile feedback (FV). Finally, subjects reported the perceived realism through a questionnaire. Results are reported, among others, in Fig. 10. Both show that

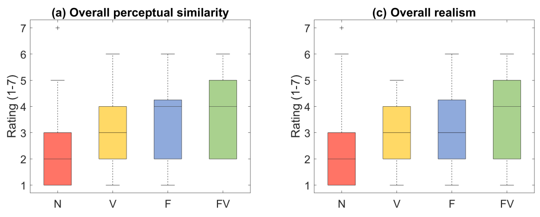


Figure 10: *Experiment 2: Overall perceptual similarity and realism.*

the combined force and vibrotactile feedback was rated as significantly more realistic. This conclusion suggests that expanding the feedback of Keytar with vibrations in both its keyboard and stylus controllers should further increase its realism.

5. CONCLUSIONS

Keytar prototypes a point-wise physical interaction with a set of virtual strings. With relatively low software programming effort, it can model essentially any compact stringed instruments that is played using a plectrum. Ongoing work is oriented to reproduce interactions capable of taking full advantage from alternative prototypes. The lap steel guitar, for instance, could be realized by displaying the strings horizontally below the robotic arm; in parallel the keyboard, if not substituted by a continuous controller, could be used to set the final chord where the steel bar is sliding to. In the longer term, the keyboard (or any other controller) and the plectrum could convey string vibrations through proper actuation, as well as the plectrum position could be used to modulate the spectral content of tones as it happens with real guitars.

6. ACKNOWLEDGMENTS

This work is supported by NordForsk's Nordic University Hub Nordic Sound and Music Computing Network NordicSMC, project number 86892.