applied sciences

MDPI

Article

Multisensory Plucked Instrument Modeling in Unity3D: From Keytar to Accurate String Prototyping

Federico Fontana [1,*,†], Razvan Paisa [2], Roberto Ranon [1] and Stefania Serafin [2]

[1] HCI Lab, Department of Mathematics, Computer Science and Physics, University of Udine, 33100 Udine, Italy; roberto.ranon@uniud.it
[2] Multisensory Experience Lab, Aalborg University Copenhagen, 2450 Copenhagen, Denmark; rpa@create.aau.dk (R.P.); sts@create.aau.dk (S.S.)
* Correspondence: federico.fontana@uniud.it; Tel.: +39-0432-55-8432
† Current address: Via delle Scienze 206, 33100 Udine, Italy.

Received: 29 November 2019; Accepted: 18 February 2020; Published: 21 February 2020

Abstract: Keytar is a plucked guitar simulation mockup developed with Unity3D that provides auditory, visual, and haptic feedback to the player through a Phantom Omni robotic arm. Starting from a description of the implementation of the virtual instrument, we discuss our ongoing work. The ultimate goal is the creation of a set of software tools available for developing plucked instruments in Unity3D. Using such tools, sonic interaction designers can efficiently simulate plucked string prototypes and realize multisensory interactions with virtual instruments for unprecedented purposes, such as testing innovative plucked string interfaces or training machine learning algorithms with data about the dynamics of the performance, which are immediately accessible from the machine.

Keywords: stringed musical instruments; multisensory feedback; virtual reality; musical haptics

1. Introduction

Keytar is a virtual guitar, providing sensory feedback comprehensive of three modalities: visual, auditory, and haptic. This application has been previously documented concerning its initial implementation [1,2], and concerning subsequent hardware/software refinement now including keyboard-based note control [3].

As Figure 1 shows, Keytar's users pluck a string through the robotic arm using their dominant hand, meanwhile selecting notes or chords through the keyboard controller with the other hand. As the dominant hand feels the resistance and textural properties of the string actuated by the robot, the plucking force follows by a perception-and-action process which comes naturally for most musicians, and especially guitarists. The setup is completed by a screen displaying the action of the virtual plectrum against the vibrating strings, as well as by the auditory feedback consequence of string excitations.

Apart from Guitar Hero—a best-selling music software game based on guitar-like controllers [4]—several plucked instrument models have been prototyped in the form of virtual reality interfaces, sound synthesis models, or educational scenarios. Virtual Air Guitar is a touch free virtual musical instrument controlled through graspable accelerometers and camera hand tracking [5]. This instrument was shortly followed by Virtual Slide Guitar [6], whose control was realized using high-rate infrared visual capture. The air guitar concept was later generalized into a programming framework [7], also targeting specific control devices, such as the Kinect [8]. Other plucked virtual instruments include the harp [9] and kalichord [10].

Figure 1. The Keytar interface, with a robotic arm for plucking and a keyboard for selecting the notes of the virtual strings.

Augmented reality has been also used to support guitar learning [11] and rehabilitation courses where patients were invited to hear the sound of their own guitar playing actions [12].

Keytar is developed using the Unity3D environment, a currently widely-used visual programming environment for prototyping computer games and virtual reality scenarios, as has been suggested [13], advertised (https://rapsodos.ru/watch/How-to-Create-a-Piano-in-Unity-3D/), and previously used [14] to develop virtual instruments. Its environment can be provided with a library of 3D instrument models (https://assetstore.unity.com/packages/3d/props/musical-instruments-pack-20066). Yet, multisensory musical objects are not part of the standard assets that can be imported into a Unity3D project.

In this work we summarize and extend the results from previous presentations [1–3], describing how Keytar is evolving in a more general software tool allowing for prototyping of plucked string instruments inside Unity3D. In this sense, this paper is not about new digital music instruments. Rather, it offers the state of the art about the use of a game engine, in this case Unity3D, for the specific research topic. Its reading, hence, is not expected to enrich sonic interaction designers' basic knowledge about the virtual prototyping of musical instruments; conversely, it should help them understand Unity3D's programming principles and basic philosophy, and thus estimate the current pros and cons of this development environment before starting a new project.

For this reason we do not expect this work to raise interest in the long-term. Rather, our goal is to immediately stimulate further research and prototyping of digital musical instruments through the explanation of the pros and cons. Among current concerns, we will report about inaccuracies of the collision detection mechanism of the existing environment, as well as limitations in the visual rendering of rapidly oscillating objects. To this regard, the most important part of our on going work consists of the substitution of static with dynamic 3D elements forming each string.

Here, the potentially interested reader may wonder why we approach such a sophisticated framework for sonic interaction design purposes. Indeed this question animated the development of Keytar since the beginning. As this project posed a broad range of questions about the simultaneity and accuracy of the performer–instrument interaction in a multisensory context, Keytar became a tool to uncover the applicability of the Unity3D's programming methodology to address prototyping subtleties which typically appear in sonic interaction design. Through its development, we have been able to assess several factors in this methodology across a bottom-up process, and consequently report to the community.

Our conclusions are that Unity3D will soon be able to offer a competitive set of specific functions for digital music instrument designers at no excessive costs in terms of the time to learn. On the other hand, the same designers' community will have to actively be involved in the refinement, especially of the software libraries enabling the visual programming of physical interactions that are at the base of sound synthesis. As, for instance, haptics researchers are currently doing while updating the libraries enabling the robotic arm used by Keytar.

Applications of the resulting string objects include the design and virtual prototyping under Unity3D of new plucked instruments, possibly for use with intelligent agents that may perform and learn from them much more easily than by exchanging multisensory data with a physical instrument, even if it is provided with proper sensors and actuators. Whatever the envisaged application, an accurate plucked string model running within Unity3D would enable interaction designers to sketch and test musical instrument prototypes faster than with any other development environment.

Figure 2 gives an overview of the interaction design components and software libraries used to implement them. The most critical aspect related to the use of each library is listed. Together, such aspects emphasize the exploratory character of this paper and call for further work, aiming at understanding in more detail how the corresponding software libraries could work optimally together. In this respect, our investigation represents an initial effort toward this direction and, hopefully, a stimulus for other researchers to program a physical model for Unity3D capable of integrating the haptic interaction with the audio–visual animation.

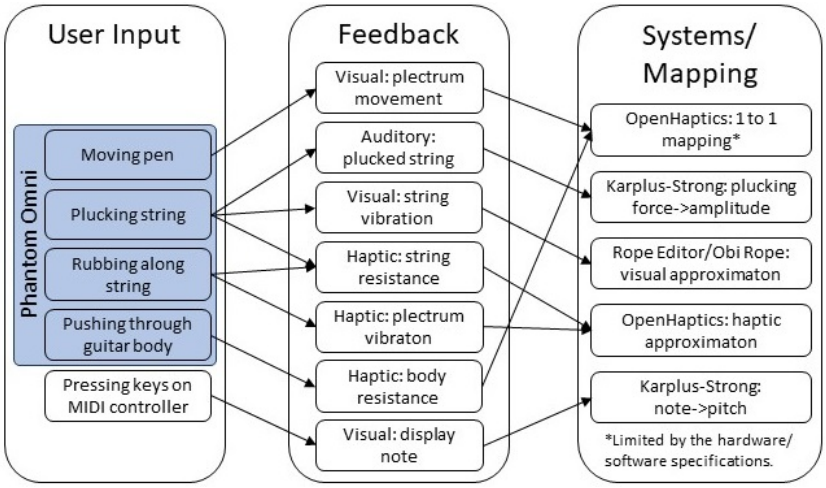

Figure 2. Overview of the interaction design components and related software libraries.

This paper addresses the design of the Keytar haptic workspace in Section 2. Then, Section 3 is devoted to the issues created by the previous workspace, and consequent modifications that were necessary in order to achieve a more accurate visual rendering and, prospectively, a more precise haptic interaction. Section 4 describes how the string sounds are synthesized, and how they can be adapted to specific project requirements using a proper chain of digital audio effects. Section 5 discusses the perspectives of the current development environment, mainly limited by the hardware at hand. Finally, Section 6 concludes the paper.

2. Haptic Feedback

Currently, many musical instrument computer interfaces consider haptic feedback as a necessary component that must be made available to the user. The idea is not new [15], as testified by several decades of development of e.g., the Cordis-Anima framework [16]. However, the relatively recent availability of affordable force and especially vibratory devices has brought to the foreground the effects that haptics have during performances [17]. Such effects have been shown to play a significant role,

for instance, on the perception of quality in pianos [18] and violins [19], and the research field has gained enough relevance to become known as Musical Haptics [20].

In our case, force and tactile feedback are enabled by a SensAble (now 3D Systems) Phantom Omni robotic arm, as shown in Figure 1. The motors of this device exert limited force (up to 3.3 N) and can actuate rotation and lift. On the other hand the arm movements are extremely precise (tracking accuracy equal to about 0.055 mm), silent and prompt, as the mechanics have been designed to simulate materials that are not too stiff. For this reason the Phantom Omni proved ideal when simulating string plucking with a plectrum. In addition to responding with realistic forces, the motors of the Phantom Omni are able to follow little changes in the contact point position. In this way they enable the sonic interaction designer to define textural motifs over the strings that will be plucked.

At this point, if the user shifts the plucking point, as guitarists, for instance, do when they rub the plectrum along a wound string, then he/she will have the tactile feeling of scraping the string, depending on its coating, with a net effect on the perceived realism of the interaction. This feature of the robotic arm has been used in Keytar since its first version. In parallel to feedback, the Phantom Omni has good characteristics as an input interface for capturing the plucked interactions mediated by a plectrum. The device, in fact, records the tip position and angle of its stylus (visible in Figure 1) with a 1 kHz sampling rate, that allows the capture of the plectrum position with an accuracy of 1 mm when the performer's hand plucks the string at 10 m/s, which is far above reported hand strumming velocities—normally below 1 m/s [21].

Haptic Workspace

The robotic arm is interfaced to the virtual scene in Unity3D through the *haptic workspace*. This workspace contains the tangible properties of all the objects in the scene. In the limit, such objects can be set to be invisible, that is, absent in the visual space but reactive in terms of haptic feedback, or conversely, visible, however absent in the haptic workspace. Unfortunately, complex haptic workspaces enabling contacts between 3D objects which are not primitive in Unity3D often encumber the simulations, causing occasional crashes of the application, especially if built for slower machines using the Haptic Plugin for Geomagic OpenHaptics—see the end of this section.

Another consequence of object complexity is that, even in the absence of a system crash, the plectrum-string contact can go unnoticed by the system due to the real-time constraint imposed on the collision detector for returning a result. This is a well known problem in game development, sometimes referred to by Unity3D developers as the *Bullet through paper problem*. (https://answers.unity.com/questions/176953/thin-objects-can-fall-through-the-ground.html).

These limitations motivated the design of a simple physical contact, by providing the haptic workspace with a little sphere corresponding to the tip of a larger visual plectrum (see Figure 3): The former, responsible for making contact with the strings, was instantiated as a *Sphere* object having a minimum (i.e., almost point-wise) diameter. In parallel, each haptic string was made of a chain of *cylinder* objects active both in the haptic and visual space. Both spheres and cylinders are primitive in Unity3D and possess parameters of diameter, position, and length that were set along with their haptic properties during previous tests of the system, also involving guitar and bass players [1].

An obvious downside of this simplification is that the sphere cannot provide a distributed contact point, nor allow a performer's freedom to rotate the plectrum as a larger triangular object would have been able to. Moreover, the plectrum could visually pass through the strings as soon as its tip was incidentally moved below them instead of colliding against their surface, even diving into the fretboard if the user continued to pull down the arm of the robot. Even worse, recovering from this unrealistic situation was problematic as the plectrum easily collided against the strings on its way back, making it difficult for the user to restore a correct interaction.

This issue was solved by inserting an invisible haptic surface immediately below the strings, opposing maximum stiffness against penetration. The inclusion of this additional object overall resulted in the haptic workspace shown in Figure 3.

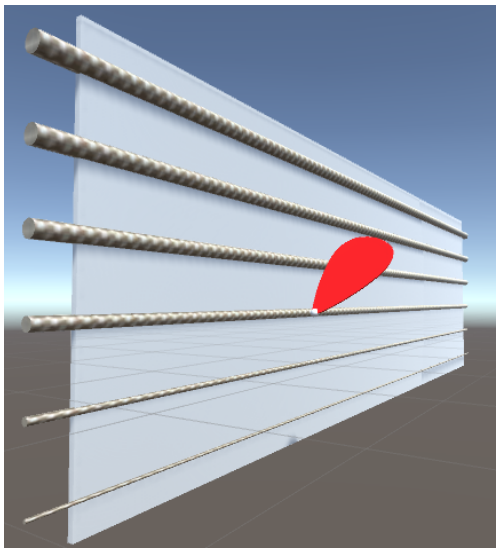

Figure 3. The haptic workspace in Keytar. The colliding sphere is visible as a white tip on the edge of the plectrum.

A more subtle visuo-haptic mismatch manifested when the haptic edge of the plectrum touched, but did not penetrate a string enough: In this case, the haptic objects did interact, but did not collide deeply enough to elicit sufficient force feedback to the user. This problem was solved by wrapping each string with an invisible, yet touchable, layer, which was set to be as thick as the radius of the haptic sphere forming the plectrum. The layer is shown in Figure 4. This ensured that users, while plucking the visible string, were moving the stylus on a position providing robust contact between the plectrum and the string.

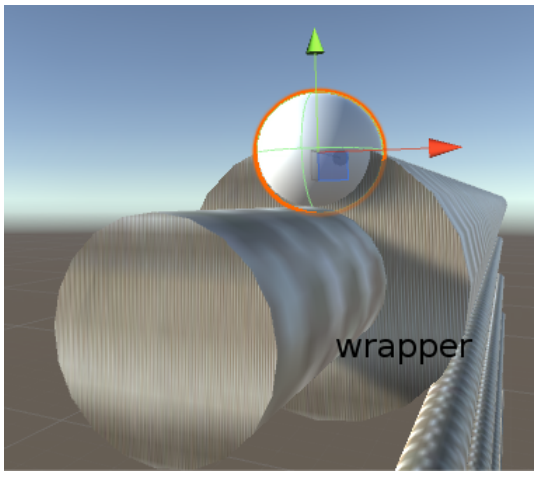

Figure 4. Mismatching positions of the visual and haptic collision point. Wrapping with a touchable layer.

On top of providing force feedback to the plectrum, due to their peculiar design made by alternating small cylinders of two different diameters, the strings conveyed tiny discountinuities that the robotic arm was able to render when the stylus was slid to scrape the plectrum along a string. This feedback provided a vibrotactile effect that differed among strings by their size (depending on the diameter of the cylinders) and texture (depending on the length and diameter differences between adjacent cylinders). In any case, the strings were static objects, in the sense that their position did not change in time, as opposed to the plectrum. Furthermore, there was no vibrotactile feedback from the string to the plectrum, as the string vibration was only rendered visually.

On a brief technical note, Unity3D objects that need to be placed in the haptic workspace must be interfaced to the classes *RigidBody* and *Collider*. These classes provide methods for the real-time detection and management of collisions. Such objects become visible to the robotic arm if they are furthermore typed as *Touchable*. Once interfaced also to this class, they in fact have access to Unity3D's Haptic Plugin for Geomagic OpenHaptics resources, developed for Windows and (in beta version) for Linux by the Glasgow School of Art's Digital Design Studio. The interested reader is referred to the respective asset (https://assetstore.unity.com/packages/essentials/tutorial-projects/unity-5-haptic-plugin-for-geomagic-openhaptics-3-3-hlapi-hdapi-34393) for Unity3D v.5 and related information.

3. Visual Feedback

As we have seen, the only dynamic object populating the haptic workspace is the plectrum. On the one hand, this configuration is satisfactory as guitarists and other stringed instrument players do not experience positional changes of the strings while playing their instruments, and the compliance of the robotic arm in parallel proved suitable to convey a credible sense of string elasticity through the virtual plectrum. On the other hand, strings that do not visually vibrate after being plucked are unrealistic. Similarly to the haptic–visual case, in the audio–visual case we achieved sufficient realism by visually animating the strings, yet with no relationship between the sound produced by the Karpulus–Strong algorithm and the visual vibration of the strings. As previously said, in Keytar the visual scene was organized independently of the haptic workspace, so in principle it would be sufficient to design a realistic graphic rendering of the vibrating strings to add consistent visual feedback to the overall scenario.

The experience with Keytar showed that accurate vibrating strings are not straightforwardly rendered in Unity3D. While plectrum movements and rotations were directly reproduced from the standard *camera* viewpoint in the visual scene, strings visually vibrated thanks to the *Animator* interface, whose methods implement Unity 3D's Mecanim. (https://docs.unity3d.com/Manual/AnimationOverview.html). This animation system concatenates basic *Animation* objects along time, together realizing the visual flow visible from the *Camera* object. *Animation* objects were designed to model transversal oscillations with specific frequency and time decay, in ways that few such objects could be used to animate each string with an overall low computational effort. In practice, each collision triggered a script determining the initial vibration amplitude. From this moment on, vibrations were rendered by an *Animator* method that destroyed a visual string while cloning it into a shifted instance, until the original string returned to rest or, conversely, was triggered again by a new collision.

In spite of its efficiency and versatility, this graphic rendering technique was far from producing accurate results. Its main artefact consisted of a purely transversal motion of the strings as they were all rigid bodies. The the vibration frequency also needed to be higher. The most promising solution was that of redefining the design approach completely, through the use of a dynamic model. Furthermore, if a dynamic model was correctly computed in the visual scene, then an attempt could be made to export it also to the haptic workspace, with potential advantages in terms of the accuracy of the somatosensory feedback and coherence of the multisensory percept. In other words, the new approach could also be used as a multimodal platform rendering string vibrations.

With this goal in mind, the *Animator* interface was substituted in the Keytar software architecture with the Ultimate Rope Editor asset (https://assetstore.unity.com/packages/tools/physics/ultimate-rope-editor-7279), available from the Unity3D Marketplace. This asset provides the dynamic generation of strings of variable length and the number of links and nodes. Once imported into a project, the Ultimate Rope Editor allows for editing several pseudo-realistic physical parameters, such as string tension, string weight, and breaking force.

Due to its flexibility and the dynamic processing of string vibrations, this asset provided far greater visual accuracy. Its feedback is appreciable from Figure 5. Though, such an improvement came

at costs that are listed in Section 5, where the use of the Ultimate Rope Editor as a dynamic model integrating together the visual and haptic behavior of the vibrating strings is discussed.

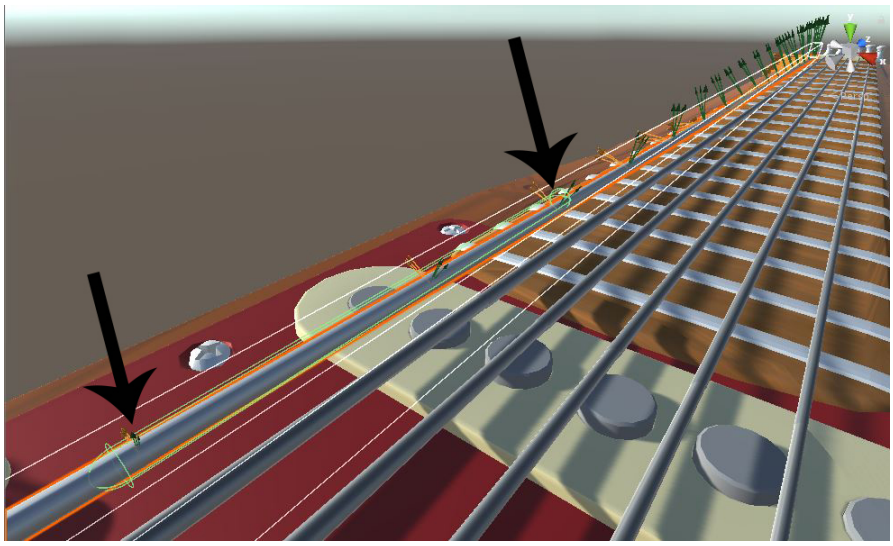

Figure 5. Snapshot of the visual feedback in Keytar using Ultimate Rope Editor. The active region of the collision can be seen bounded by green circles surrounding the first string, each pointed by a black arrow. Small green and red arrows depict the horizontal and vertical components of tangential force vectors exerted by the active string elements.

4. Auditory Feedback

In the visual scene, the amplitude of each vibrating string was computed starting from the spatial coordinates (x, y, z) of the contact point, immediately before (b) and after (a) every collision. These coordinates are accessible through the Haptic Plugin at runtime, and provided a measure of the initial amplitude as each string was shifted when its oscillation started. This measure was used only to excite the algorithm, and not to change its parameters based on the plucking position. Nevertheless, the same data allowed for an estimation of the velocity, computed as the distance between such two points divided by the haptic frame rate (1 ms, see Section 2) readable from the static variable *Time.deltaTime*:

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

The velocity was then used to determine the intensity of the acoustic vibration in each string.

In Unity3D, the sound sources and listening points are respectively defined by *AudioSource* and *AudioListener* objects. Keytar contained six sound sources, one for each string, and a default audio listener corresponding to the *Camera* object. Each string was sonified by means of a Karplus–Strong [22] sound synthesis algorithm, available as an asset (https://github.com/mrmikejones/KarplusStrong) of Unity3D. Concerning the algorithm parameters, tone fundamental frequency, and sound intensity were set for every string at runtime through a keyboard-based Musical Instrument Digital Interface (MIDI) control.

The keyboard, visible in Figure 1, was in fact connected to the sound synthesizer, thanks to Keijiro Takahashi's MIDI Jack open project (https://github.com/keijiro/MidiJack) for Unity3D. In spite of its low-latency and moderate computational cost, Karplus–Strong is known to produce pure string oscillations whose sound is still far from that of a guitar. In order to improve such sounds, a series of built-in digital audio effects were cascaded along the main channel of the *AudioMixer*. The resulting chain, shown in Figure 6, followed a common "Less to More" electric guitar pedalboard configuration:

1. Attenuation is a default effect found on all of Unity's mixing channels. It sets the intial gain of the channel.
2. Distortion, with a level set to 0.82, is the first real effect in the chain that simulates a high gain drive pedal, like the BOSS DS-1. This effect provides a richer sound for the following delay module.
3. Flanger is the fastest temporal effect in the path, thus it is found early in the chain. It is set to provide some pitch modulation to the otherwise too static sound synthesized by the Karplus–Strong algorithm. The modulation rate was set at 0.1 Hz, the depth was set at 100% and the mix was set to be 70% wet.
4. Chorus makes the sound processed by the flanger fatter. The delay was set to 42.5 ms, and the time modulation oscillates at 0.8 Hz with a depth equal to 30%. There is no feedback in the chorus, however the effect provides a multitap mix control whose values were all set to 0.77. The level of dry mix was set to maximum, to ensure that some peculiar aspects of the Karplus–Strong sound were preserved.
5. A compressor is found next, taming occasionally strong attack levels caused by the Karplus–Strong algorithm. Its threshold was set to be rather high, at 5.8 dB, with a moderately fast attack of 40 ms and a medium release of 190 ms. In addition to smoothing the attack, the compressor manages to reduce the gain by 6.2 dB. This reduction results in a smaller dynamic range, however, closer to a high gain guitar tone sound. As for any digital chain, the noise floor was not increased by the compressor nor was any extra noise introduced by the other effects.
6. Distortion 2 is placed at this point to simulate the natural drive of guitar amplifiers. The level was set to a relatively low value, i.e., 0.3, as most of the harmonic enrichment had been already ensured by the previous effects.
7. A parametric equalizer (ParamEQ) is a series of peaking and notch filters which is usually put at the end of the chain. The equalizer was tuned to attenuate the spectral power around 3.5 kHz in ways to reduce some of the residual metallic timbre of the sound. The gain was set to 0.41 and the selectivity amounted to 0.23 octaves. This selectivity worked well for all strings, therefore it is reasonable to assume that the unwanted gain in the respective bands was a by-product of the effect chain.

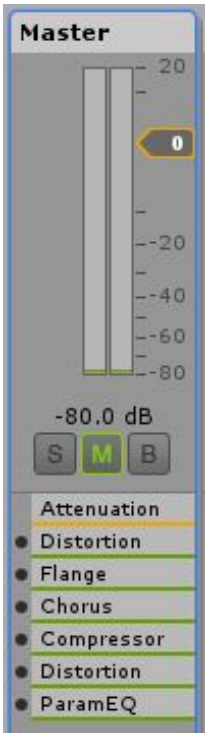

Figure 6. Virtual pedalboard.

Such effect levels and parameters were set by listening to each string, using subjective criteria like consistency among string sounds and similarity to a high gain guitar tone.

Concerning note control, Keytar featured three different playing modes that can be selected through the stylus buttons: solo, harmonic major, and harmonic minor. This feature is being maintained in the current development. Any switch of these buttons called 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—but not below the lowest note the excited string can produce, however. The harmonic modes instead launch several instances of the same algorithm, by building major and minor chords based on the last key press on the controller. This event dictates the fundamental note of the 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 that the robotic arm makes while reproducing the string textures. This effect is to be kept as well. As can be seen from Figure 3, each string results from alternating cylinders of two different diameters. The resulting surface geometry thus 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, creating a substantial improvement of the overall realism of the interaction.

5. Ongoing Work

As anticipated at the end of Section 3, an attempt to integrate the visual and haptic workspaces together under the control of a single dynamic model was made using the Ultimate Rope Editor. Keeping the conservative approach used with the previous string models in an effort to reduce the computational load of the haptic workspace, the collision detection region was limited to the area around the pickups. The locus of interaction was realized by a *Capsule* object, by attaching a collider to a node between two rope segments, as shown in Figure 7.

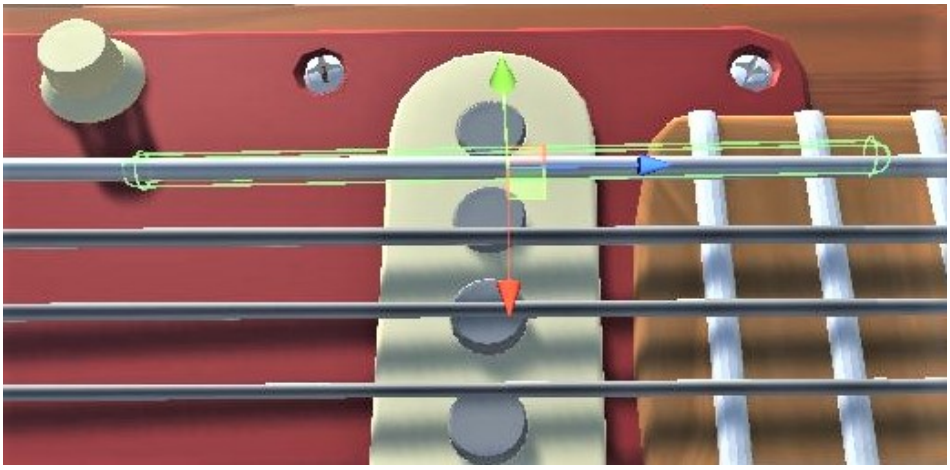

Figure 7. A *Capsule* collider attached to a string. The green, red, and blue arrows depict the orientation of the capsule object.

The diameter of the collider was matched to the diameter of the contact sphere on the plectrum, as shown in Figure 4.

The visual result was the desired one, but three problems arose in the haptic workspace using this implementation.

1. The string dynamics are not unconditionally stable when realistic parameters (i.e., low weight and high tension) are set, causing chaotic oscillations of the string links that occurred rarely.
2. An even more significant problem was the inconsistency of the haptic feedback. This behavior, in part, happened because the haptic collider was applied on the string links, which are actually free to micro-oscillate and rotate in the 3D space due to the nature of the dynamic model. These movements, visible in Figure 5 with green arrows showing the rotation of the force vectors along space, are translated to the plectrum, which rotate around the colliding sphere, thus not resisting movement. This shortcoming was emphasised by the insufficient frame rate of the Unity3D collision detection system, again causing the plectrum to occasionally move beyond the string instead of plucking against it. This issue has been recently addressed in newer versions of Unity, which now detect collisions through a callback procedure. However, the new collision detection system is yet to be implemented in Keytar.
3. The Ultimate Rope Editor is especially demanding in terms of computational effort. This increases the probability of the application to crash proportionally with the rendering accuracy set for the visual feedback.

An alternative implementation of dynamic string motion has undergone preliminary testing by the authors, leading to promising results. This implementation makes use of the Obi Rope physics engine (https://assetstore.unity.com/packages/tools/physics/obi-rope-55579), which is built on Obi *Particles* instead of Unity3D rigid bodies and links. This approach improves the customization and stability, and should lead to substantial improvements, when used in parallel with an official Application Programming Interface (API) released for Unity3D by 3D Systems, current owner of the Phantom Omni technology. This API is based on the newer version, 3.5, of OpenHaptics, and comes bundled with updated drivers for Windows 10, Mac, and Linux.

On top of the increased stability, the new API gives access to additional material parameters, such as viscosity and smoothness. The availability of such parameters promises to give the designer control of the textural parameters of the virtual strings, which is so far an exclusive domain of the arm motor characteristics. A reliable software building of the newly released Obi Rope and OpenHaptics in the Keytar architecture is the object of onging work.

6. Conclusions

If the new physics engine and API for the robotic arm provide the promised enhancements in comparison to the previous software libraries, Keytar will provide interaction designers with a development environment to visually edit, for the first time, most features of a plucked string instrument, including the size, position, plucking points, and intonation of a string set. In parallel, a wide palette of parameters will be made available to design not only the sound, but also the tactile feeling of such strings. All of these features will be in easy reach for those designers who have enough knowledge of Unity3D— a development environment that has gained popularity especially for its rich, user friendly and easy to learn visual programming features.

The multisensory feedback conveyed by Keytar has already been judged to be especially satisfactory in both controlled experimental settings [23] and informal showcasing situations [3]. Its prospectively more stable behaviour and wider accessibility to the contact parameters make Keytar an ideal candidate for the virtual testing of novel plucked string interactions, and for the interactive generation of music performance data. Such data could be used, for instance, as an initial platform for the comparative analysis of touch and sounds using different sets of virtual strings. The tuning of such strings could be made by engaging musicians in interactive tasks, where they manipulate physical parameters based on subjective haptic and auditory sensations.

Author Contributions: Conceptualization, F.F. and R.P.; software, R.P.; resources, F.F., R.R. and S.S.; writing—original draft preparation, F.F.; writing—review and editing, F.F., R.P., R.R. and S.S.; project administration, F.F.; funding acquisition, S.S. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by NordForsk grant number 86892 NordicSMC - Nordic Sound and Music Computing Network.

Acknowledgments: Andrea Passalenti provided the original conceptualization and software of Keytar.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Passalenti, A.; Fontana, F. Haptic interaction with guitar and bass virtual strings. In Proceedings of the 15th Sound and Music Computing Conference (SMC 2018), Limassol, Cyprus, 4–7 July 2018; pp. 427–432.
2. Passalenti, A.; Paisa, R.; Nilsson, N.C.; Andersson, N.S.; Fontana, F.; Nordahl, R.; Serafin, S. No Strings Attached: Force and Vibrotactile Feedback in a Guitar Simulation. In Proceedings of the 16th Sound and Music Computing Conference (SMC 2019), Málaga, Spain, 28–31 May 2019; pp. 210–216.
3. Fontana, F.; Passalenti, A.; Serafin, S.; Paisa, R. Keytar: Melodic control of multisensory feedback from virtual strings. In Proceedings of the International Conference on Digital Audio Effects, (DAFx-19), Birmingham, UK, 2–6 September 2019.
4. Miller, K. Schizophonic performance: Guitar hero, rock band, and virtual virtuosity. *J. Soc. Am. Music* **2009**, *3*, 395–429. [CrossRef]
5. Karjalainen, M.; Mäki-Patola, T.; Kanerva, A.; Huovilainen, A. Virtual air guitar. *J. Audio Eng. Soc.* **2006**, *54*, 964–980.
6. Pakarinen, J.; Puputti, T.; Välimäki, V. Virtual slide guitar. *Comput. Music J.* **2008**, *32*, 42–54. [CrossRef]
7. Figueiredo, L.S.; Teixeira, J.M.X.N.; Cavalcanti, A.S.; Teichrieb, V.; Kelner, J. An open-source framework for air guitar games. In Proceedings of the VIII Brazilian Symposium on Games and Digital Entertainment, Rio de Janeiro, Brazil, 8–10 October 2009; pp. 74–82.
8. Hsu, M.; Kumara, W.G.C.W.; Shih, T.K.; Cheng, Z. Spider King: Virtual musical instruments based on Microsoft Kinect. In Proceedings of the International Joint Conference on Awareness Science and Technology Ubi-Media Computing (iCAST 2013 UMEDIA 2013), Aizuwakamatsu, Japan, 2–4 November 2013; pp. 707–713.
9. Taylor, T.; Smith, S.; Suh, D. A virtual harp with physical string vibrations in an augmented reality environment. In Proceedings of the International Design Engineering Technical Conferences and Computers and Information in Engineering Conference (ASME), Cleveland, OH, USA, 6–9 August 2007; pp. 1123–1130.
10. Schlessinger, D.; Smith, J.O. The Kalichord: A Physically Modeled Electro-Acoustic Plucked String Instrument. In Proceedings of the New Interfaces for Musical Expression (NIME), Pittsburgh, PA, USA, 4–6 June 2009; pp. 98–101.
11. Liarokapis, F. Augmented Reality Scenarios for Guitar Learning. In Proceedings of the Theory and Practice of Computer Graphics (TCPG)- Eurographics UK Chapter, Canterbury, UK, 15–17 June 2005; pp. 163–170.
12. Gorman, M.; Lahav, A.; Saltzman, E.; Betke, M. A camera-based music-making tool for physical rehabilitation. *Comput. Music J.* **2007**, *31*, 39–53. [CrossRef]
13. Manzo, V.J.; Manzo, D. Game Programming Environments for Musical Interactions. *Coll. Music Symp.* **2014**, *54*. [CrossRef]
14. Zhaparov, M.; Assanov, U. Augmented reality based on Kazakh instrument "Dombyra". In Proceedings of the IEEE 8th International Conference on Application of Information and Communication Technologies (AICT), Paris, France, 20–24 July 2014; pp. 1–4.
15. Berdahl, E.; Verplank, B.; Smith, J.O.; Niemeyer, G. A Physically Intuitive Haptic Drumstick. In Proceedings of the International Computer Music Conference (ICMC), Copenhagen, Denmark, 27–31 August 2007; pp. 150–155.
16. Leonard, J.; Cadoz, C. Physical Modelling Concepts for a Collection of Multisensory Virtual Musical Instruments. In Proceedings of the New Interfaces for Musical Expression (NIME), Baton Rouge, LA, USA, 31 May–3 June 2015; pp. 150–155.
17. Miranda, E.R.; Wanderley, M.M. *New Digital Musical Instruments: Control and Interaction Beyond the Keyboard*; A-R Editions: Middleton, WI, USA, 2006.
18. Fontana, F.; Papetti, S.; Järveläinen, H.; Avanzini, F. Detection of keyboard vibrations and effects on perceived piano quality. *J. Acoust. Soc. Am.* **2017**, *142*, 2953–2967. [CrossRef] [PubMed]
19. Wollman, I.; Fritz, C.; Poitevineau, J. Influence of vibrotactile feedback on some perceptual features of violins. *J. Acoust. Soc. Am.* **2014**, *136*, 910–921. [CrossRef] [PubMed]
20. Papetti, S.; Saitis, C., Eds. *Musical Haptics*; Springer International Publishing: Berlin, Germany, 2018.

21. Romero-Ángeles, B.; Hernández-Campos, D.; Urriolagoitia-Sosa, G.; Torres-San Miguel, C.R.; Rodríguez-Martínez, R.; Martínez-Reyes, J.; Hernández-Vázquez, R.A.; Urriolagoitia-Calderón, G. Design and Manufacture of a Forearm Prosthesis by Plastic 3D Impression for a Patient with Transradial Amputation Applied for Strum of a Guitar. In *Engineering Design Applications*; Öchsner, A., Altenbach, H., Eds.; Springer International Publishing: Cham, Switzerland, 2019; pp. 97–121. [CrossRef]
22. Karplus, K.; Strong, A. Digital Synthesis of Plucked String and Drum Timbres. *Comput. Music J.* **1983**, *7*, 43–55. [CrossRef]
23. Passalenti, A.; Paisa, R.; Nilsson, N.C.; Andersson, N.S.; Fontana, F.; Nordahl, R.; Serafin, S. No Strings Attached: Force and Vibrotactile Feedback in a Virtual Guitar Simulation. In Proceedings of the IEEE Conference on Virtual Reality and 3D User Interfaces (VR), Atlanta, GA, USA, 22–26 March 2019; pp. 1116–1117. [CrossRef]