A FAUST-BUILT MOBILE VOCODER INSTRUMENT

Riccardo RUSSO (riccardo.russo19@unibo.it) [1,2] and **Daniel OVERHOLT** (dano@create.aau.dk) [1]

[1]**Aalborg University**, Copenhagen, Denmark
[2]**University of Bologna**, Bologna, Italy

ABSTRACT

The growth of increasingly powerful mobiles devices and their ubiquity opens up more and more possibilities in the creation of New Interfaces for Musical Expression (NIMEs). However, since smartphones were not conceived for musical purposes, they are affected by some limitations.

This work aims to develop a mobile version of a vocoder using the Faust programming language, in order to test the limits and opportunities offered by smartphones in creating a portable version of such an old musical effect. Both a custom app and purpose-designed phone case prototype were developed. The vocoder app presents a clear reconstruction of the words, via the quite pleasant and well-known timbre. However, some difficulties were encountered in the development process. In particular, some mobile devices are not powerful enough to handle a high level of polyphony.

1. INTRODUCTION

The last fifteen years have seen a sharp increase in mobile phones computational power, to the point that what we now carry in our pocket is essentially a small computer, capable of carrying out many of the tasks that have been traditionally performed by laptops. Concerning the audio field, many DSP algorithms which require a high computational power can now run on relatively inexpensive devices. Besides greater processing power, smartphones nowadays include a plethora of sensors for interfacing with the outside world, which opens up many mapping possibilities. Furthermore, the ubiquity of these devices provides a wide user base. These features together make smartphones suitable for the development of more and more complex New Interfaces for Musical Expression (NIMEs) [1].

The characteristics mentioned above are appealing; nevertheless, many limitations need to be considered when developing a NIME on mobile devices; these can be summarised by the phrase: "smartphones are not conceived for musical purposes". This statement reflects itself in both hardware and software limitations [2, 3]. From the hardware side, the main problem lays in the touchscreen being the primary form of input. Despite being easy to use, this technology provides almost no tactile feedback response, a crucial factor in the development of a musical interface [4]. Moreover, touchscreens usually do not capture pressure information. Apple alone tried to overcome this limitation, by developing the 3D Touch technology and introducing it in its mobile devices. Nevertheless, this feature was soon abandoned. This fact shows that more sophisticated interfaces are not of commercial interest [5]. Smartphone sensors such as accelerometers and gyroscopes can work as input devices, however, since they are not designed for accurate musical mapping, the sensor data usually needs to undergo some pre-processing before being employed for musical purposes. Finally, despite the latest smartphones offering high-quality ADCs and DACs, external input/output interfaces are usually limited to mini-jack and Bluetooth connections. From the software side, the two major operating systems, iOS and Android, present different limitations. The first always provided great performance for audio, and vast development support; however, prototyping and developing on Apple machines usually requires permission from Apple, which might discourage amateur developers. Android, on the other hand, traditionally put less importance on audio dsp capabilities, focusing on other tasks; however, it allows for a development process with no restrictions.

1.1 Smartphone as a Portable Instrument

Despite the aforementioned limitations, several works have investigated the opportunities offered by smartphones as platforms for the development of NIMEs ever since their initial spreading [1], sometimes trying to extend the devices' functionalities. [6].

An early work by Geiger [7] studied the possibilities provided by the touchscreen itself as a controller. In this paper, he first mentions how the touchscreen allows to overcome the classic WIMP (Windows, Icon, Menu, Pointer) interface, which has proven to be difficult to handle in live musical performance. Secondly, he defines a number of interaction principles for the development of instruments interfaces with touchscreen devices. According to Geiger, this interface should:

- Be one piece, and not a collection of controllers.
- Be easily portable and usable in different social contexts.
- Have an interface which maximises control and gives immediate feedback, providing control not

only on the note and volume, but also at the sound processing level.

- Be learnable and master-able, giving the player direct feedback on their progress.

These bullet points are also in accordance with the evaluation principles later provided by O'Modhrain [8].

The Stanford Mobile Phone Orchestra (MoPhO) [9] not only showed that smartphone instruments are suitable for social and collective playing environments such as an orchestra, but it proved that they can be interesting means of music production even when extended with external devices. In fact, in its 2010 evolution, players were provided with speaker gloves in order to provide amplification while keeping focus on portability.

With the growth of smartphone popularity, many musical apps were developed, introducing new ideas for touchscreen usage as a music controller. Bebot [1] is a mobile phone synth with an interface specifically designed for touchscreen devices. Inside Bebot, virtual knobs, sliders, and even keys leave space for controllers that react to the fingers movements and amount of fingers present on the keyboard. More recently, Electrospit [2] won the 2020 Guthman competition for newest and greatest ideas in music. It consists of a wearable talk-box which can be interfaced with any external synth, or with a companion app provided by the company, downloadable from the internet. These apps and devices show how much importance portability has in new musical interfaces design. In fact, not only did Electrospit transform a fairly old effect like the talk-box into a portable version, but it developed an app in order to extend portability even further. This way, Electrospit only needs the wearable device and a smartphone to be played.

In this research, a vocoder was developed in the form of an Android app, by making use of the Faust programming language. The goal was to test the portability opportunities provided by another classic musical effect that usually comes in the form of keyboard synths. The app was developed according to the Geiger principles mentioned above. Therefore, the instrument was thought to be: made of one piece, highly controllable and learnable. By taking inspiration from the MoPho, and following the work by Michon et al. [2], the vocoder was augmented with the design of a case prototype which could, at the same time, provide amplification and extend playability.

The rest of the paper is organised as follows: Section 2 briefly illustrates how a vocoder works, in Section 3 the development details are presented, Section 4 illustrates a possible evaluation protocol, and Section 5 concludes the paper.

2. THE VOCODER

The vocoder, invented in 1939 by Dudley [10], is probably the oldest vocal synthesis technique ever developed. In this method, the incoming voice signal, the *modulator*, is processed by a filter-bank which measures the gain of each frequency band. Another signal, the *carrier*, usually a broadband signal such as white noise or a sawtooth wave, is then processed by another filter-bank with the same characteristics of the first one. The gain of the filters from the second filter-bank is set using the coefficients measured in the first one. This way, the frequency content of the carrier is modulated and becomes similar to the original signal. The vocoder was an early attempt of compressing and encrypting speech information. In fact, this technique only requires to store filters coefficients instead of raw audio data. The initial idea was to apply a cipher algorithm to the coefficients and send them to a receiver station, which could decrypt them and restore the original message. Even though it has rarely been used for its original function, this algorithm became very popular in the musical field, and have been extensively used by musicians such as Earth Wind and Fire or Daft Punk. Later, the vocoder was extended by Flanagan, who formalised the Phase Vocoder [11]. This version employs the Fourier transform to split and analyse the modulator signal, thus achieving more detail and allowing to keep information on the phase. The schematic of a classic vocoder is illustrated in Figure 1.

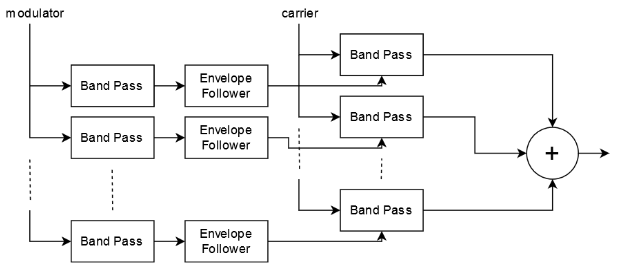

Figure 1. Schematic of a standard vocoder.

3. IMPLEMENTATION

3.1 DSP Algorithm

The vocoder algorithm was implemented using Faust [12]: a high-level, domain-specific, functional programming language, with a strong focus on the development of digital signal processing algorithms for sound and music. Faust code can be compiled with the Faust compiler, in order to directly generate standalone audio applications or plugins in different formats, or translate it to other languages such as C++, Java and others. In particular, Faust can be used to to directly build an Android app, for which the .apk file can be downloaded from the Faust database with a QR code, thus bypassing the usual development process (this is one of the reasons why Faust was chosen for this work). In Faust, data is treated as a signal stream and processed using mathematical functions. Different signals can be routed together using composition operation in a block-diagram fashion. Faust works both with a local compiler and an online one, the latter was used for this project.

Currently, Faust does not allow FFTs to be performed; therefore, the classic version of the vocoder was imple-

[1] www.normalware.com
[2] www.electrospit.com

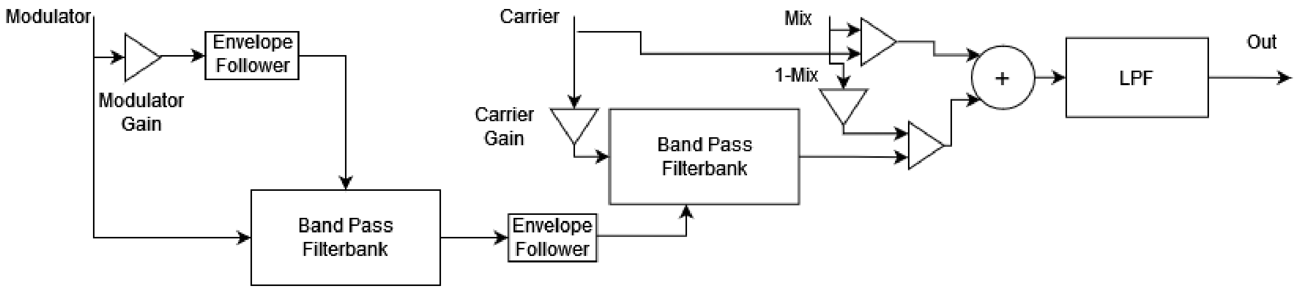

Figure 2. Schematic of the algorithm.

mented in this work. Figure 2 illustrates the structure of the developed algorithm. The modulator is obtained by capturing the user's voice with a microphone. Then the signal is split: one branch is analysed by a filter-bank made of 74 resonant band-pass filters, this number was chosen to balance the analysis capabilities with the computational power requested by the algorithm. The other branch is multiplied by a gain, controlled by the user, and analysed by and envelope follower. The filters and envelope follower used are included in the Faust libraries. The signal provided by the envelope follower is used to control the gain of the analysis filters, in order to avoid spurious sounds being outputted from the vocoder when nobody is speaking into the microphone. This way the gain of the filters is "opened" only when needed and it is possible to control the "sensitivity" of the filter bank by changing the modulator gain. Fast attack and release times were chosen for the envelope follower, with values of respectively 0.1 ms and 5 ms, in order to obtain a fast response of the vocoder. The signal output from each band of the filter-bank is analyzed by an envelope follower with the same characteristics of the first one. This signal is used to control the gain of the filters inside another filter-bank (with the same characteristics as the first one), which processes the carrier signal. The latter is previously multiplied by a user-controlled gain, which provides a general volume control for the vocoder. The carrier signal is made of a blend of a square wave and a sawtooth wave. A tone control, made with a slider, allows to play either the square wave, the sawtooth, or a mix of those. The latter contains both even and odd harmonics of the fundamental, allowing for a more accurate reconstruction of the voice. The square wave was added, even though it contains only the odd harmonics, in order to increase expressiveness, allowing to perform variations on the timbre. Another slider controls the dry/wet parameter of the vocoder: when the signal is completely dry, only the carrier is audible, and the app becomes essentially a synth. On the contrary, when the signal is completely wet, only the vocoder signal is present, meaning that playing a key without speaking into the microphone will not produce any sound. Finally, the sound is processed by a low pass filter in order to increase control over the timbre. The filter employed is a 3-pole Butterworth LPF, as included in the Faust libraries.

The overall sound is quite pleasing as vocoders can be, and the app works without significant latency as developed. In addition, voice synthesis performs well and spoken words are recognisable. However, the algorithm does not allow for a high level of timbre customisation, with the controls being limited to the choice of a sawtooth wave or a square wave, and the low pass filter cutoff frequency. As a matter of fact, the main focus of this work is to investigate the capabilities of the vocoder as a mobile instrument, and the addition of a full synth engine is left for future work. Moreover, as it will be seen in the next paragraph, the Faust framework employed makes it difficult to include many controls on the GUI. On some smartphones it was noticed that, if more than two notes are played simultaneously, the sound loses quality and becomes detuned. This is due to the device not being powerful enough to carry out all the computation. This issue might be solved by optimizing Faust code at C++ level; nevertheless, further investigation needs to be done.

3.2 Interface - Mapping

In order to develop the GUI, the Faust Smartkeyboard UI [3] tool was used. This framework allows to bypass the standard Faust UI and to implement a wide range of controllers optimised for touchscreens such as x/y pads, sliders, smart keys. In addition, the interface offers the possibility to easily access data from the smartphone accelerometers and gyroscopes. Despite the user friendliness, the framework presents some disadvantages over traditional GUI development tools. For instance, the aspect of the graphic elements of the UI cannot be changed from the default one; as such, the GUI customization is limited.

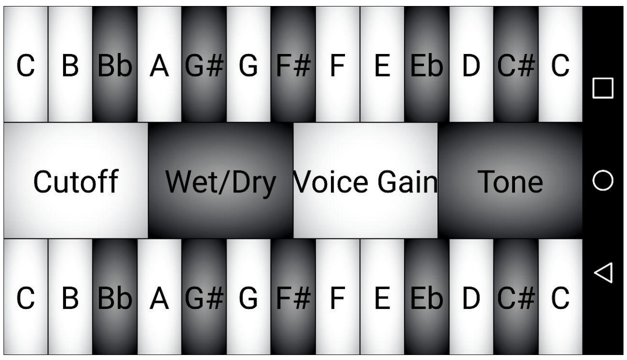
C B Bb A G# G F# F E Eb D C# C
Cutoff
Wet/Dry
Voice Gain
Tone
C B Bb A G# G F# F E Eb D C# C

Figure 3. Application interface.

Two traditional looking keyboards control the carrier fre-

[3] ccrma.stanford.edu/ rmichon/smartKeyboard

quency, namely, the note. The choice of implementing this interface, instead of developing something more "touch friendly" like what is seen in Bebot, has two explanations. The first one is that Smartkeyboard is optimised for developing a keyboard-like interface, and changing this behaviour would have presented some difficulties. The second one follows Cook's statement [13]: "copying an instrument is dumb, leveraging expert technique is smart". This means that, by creating interfaces that resembles the ones of actual instruments, it is possible to take advantage of the years of practice that the performer already underwent. Since piano interfaces are quite common also among physical synthesisers and vocoders, the statement holds also in the case of this app.

Once the vocoder was developed, it was necessary to decide which parameters could be mapped to the interface, and which ones had to be hardcoded. In fact, each algorithm block could potentially provide many parameters for controlling the instrument, something which could, in theory, have increased expressiveness. Nevertheless, two main issues arose. The first is linked to the devices' sizes, in that smartphones have small screens; therefore, adding all the possible controls would have made interface elements so tiny that playing the instrument would have been virtually impossible. The second issue is more related to the playability of the instrument. As a matter of fact, when designing a NIME it is necessary to make a decision regarding the complexity of mappings, which reflects on the cognitive load on the player [4]. Some low-level parameters might need to be mapped in a way that, after being set, the player "forgets" about them, in order to focus on other aspects of playing. Both issues could be solved by creating sub-menus where to place all the "secondary" controls. However, Smartkeyboard does not allow to do that, therefore some parameters had to be set at the code level. In particular the hardcoded values were: the attack and release time of the envelope followers (respectively 20.1 and 5 ms), the number of bands (74) and the Q-factor of the filters (0.5), the highest and lowest possible frequencies of the filter cutoff (80 and 10000 Hz) and the filter sharpness. Therefore, the mapped parameters were: carrier frequency, carrier gain, carrier tone blend, modulator gain, dry/wet and filter cutoff frequency.

The developed interface is represented in Figure 3. The app is intended to be played with the bottom of the phone facing the body; in fact, the microphone is located at the bottom of the phone. Hence, the way of playing resembles the one of flutes, in a similar fashion to Wang's Ocarina [14]. The two keyboards lay at the opposite sides of the screen. This way the instrument can be played with two hands simultaneously, plus, the carrier frequency range spans two octaves. Both keyboards span one octave, the left keyboard lowest key is a C3, the right keyboard one is a C4. This note range was chosen in order to allow to play a wide repertoire, in fact, Smartkeyboard does not allow to change the keyboard frequencies once the app is built. The y position of the fingers on the keys controls the carrier gain: the nearer the finger is to the border of the screen the louder the volume is. The center keys are sliders, controlled by the x position of the finger. "Cutoff" controls the cutoff frequency of the low pass filter: the nearer to the bottom of the phone, the higher the cutoff frequency. "Dry/Wet" controls the dry/wet parameter: the nearer to the bottom, the dryer the sound. "Voice Gain" controls the modulator gain: closer to the bottom means more gain. Finally, "Tone" controls the blend between square wave and sawtooth wave: near to the bottom of the phone means full square wave, far from it means full sawtooth wave. The smartphone x-axis accelerometer is mapped to the frequency, so that strongly shaking the phone will produce a vibrato effect. It was thought that mapping more complicated gestures to the accelerometers would have affected the overall playability.

3.3 Case Design

The vocoder naturally suffers from feedback issues. In fact, if the output sound is caught by the microphone, the instrument will self-amplify itself to the point that it will become unusable until the output sound is muted. This problem particularly affects this app; indeed, in most smartphones the speaker is located near to the microphone. For this reason, either an external speaker or a microphone are needed for playing the instrument. However, this means that, in this configuration, the app alone is not compliant with the first one of Geiger's principles mentioned in section 1. Moreover, it was noticed that playing with two hands simultaneously was not very comfortable, as one of them was busy holding up the phone, an issue also encountered in [2].

In order to overcome both issues, a case was designed, intended to hold the smartphone, increase playability, provide amplification, limit feedback issues and create a compact, portable instrument. A cardboard prototype of the case is visible in figures 4 and 5. The handle provides a stable support for the whole instrument, while leaving both hands' fingers free to play. The smartphone would be hold in position and secured by a vise, while the whole support could be adjusted in height by moving the support on a tiny rail placed on the handle. This way the case could be adapted for various hands sizes. The bottom box would hold an external speaker, in order to provide amplification. Moreover, as mentioned by Oh et al. [9], keeping the sound source close to the instrument (the keyboard in this case) provides a closer association between the instrument and the performer. The idea of including a speaker into the case introduces a problem in the design. In fact, there are two ways of doing so, each with pros and cons. The most straightforward solution would be to directly build the case with a speaker in it, in a similar fashion to what is done with the Gramophone [15]. This way, users could obtain a working instrument without any effort from their side. On the other hand, doing this would complicate the manufacturing stage. It would make it necessary to build or buy a speaker, good enough to be used as an instrument, to include a battery holder, amplification circuit, etc. These components cannot be 3D printed; therefore, this solution would complicate the building process, increasing costs too. A different way of solving the amplification issue would be to simply develop a speaker holder, which could be easily 3D printed, where users could place their own speaker. While, with this solution, the manufacturing process would be simplified, dealing with speakers of different sizes would become complicated. Given the considerations detailed above, the best solution would probably be a combination of the two. It could be possible to build a case with a speaker included and sell it (to cover the manufacturing cost). At the same time, the CAD projects could be released as open-source, thus allowing users to alter the design and build their own version of the case, adapted for their own speaker.

The instrument can be played either with the phone microphone or with an external one without affecting the playability. As mentioned above, the case was thought to be easily 3D-printed or assembled with a laser cutter. In particular, many of its components, for instance the phone holder, could be built starting from elements present in Mobile3D, a CAD library specifically designed for augmenting mobile devices [2]. An actual implementation of the case is planned as a future work.

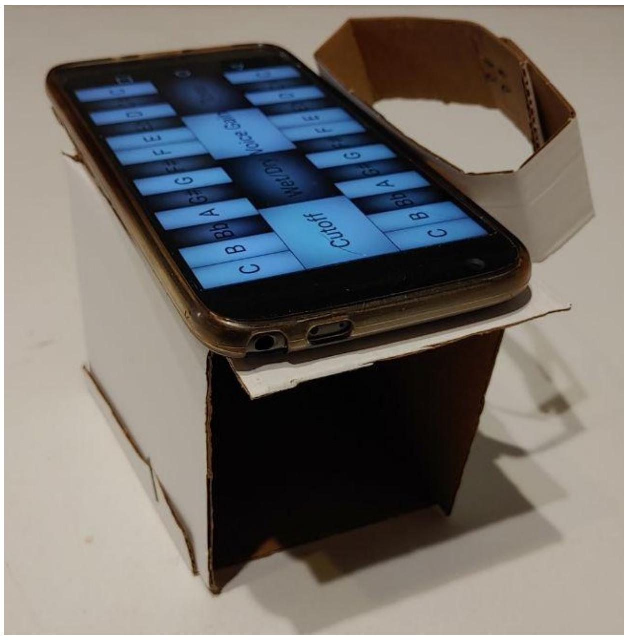

Figure 4. Case prototype. Front view. The empty box is supposed to hold an external speaker.

4. EVALUATION

Since the instrument is still at the stage of proof-of-concept, the development of an evaluation protocol is beyond the scope of this work. However, it is possible to state few possible evaluation goals. Following O'Modhrain work [8], there are different point of views when evaluating a NIME. One of the key factors in giving the audience an enjoyable performance with a NIME is to provide visual cues linking cause and effect. Since this instrument is essentially a digital and portable version of an already existing one, audience perception of virtuosity should not change. The manufacturer point of view has been already

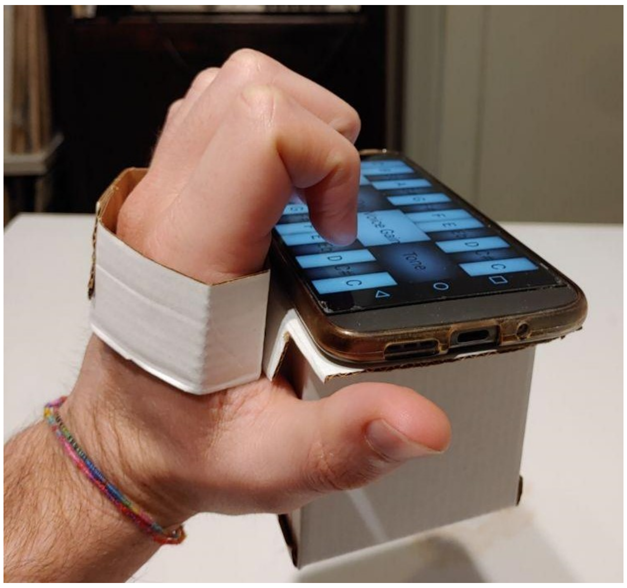

Figure 5. Case prototype. Rear view. This image provides a demonstration of how the handle works.

discussed in the previous section. In fact, the only physical device to be constructed is the case, while users would be able to simply download the app on their smartphones. This is one of the advantages of developing NIMEs on smartphones: players already possess most of the hardware in their pocket. Nevertheless, whatever building solution is chosen from the ones previously discussed, the construction of the case remains very convenient, as most of its parts can be easily 3D printed or assembled with parts made on a laser cutter. The most important point of view in the evaluation of this instrument is undoubtedly the performer's. The aim of this work was to recreate an already existing instrument in a more compact, cheap and portable form without dramatically affecting playability. For this reason, it is important that the players would be able to achieve a decent level of mastery. A straightforward way to test this aspect might be to ask performers to play classic vocoder songs.

5. CONCLUSION

An Android vocoder app has been developed and presented, along with a case prototype. The instrument performs well, producing the pleasant well-known vocoder sound, and correctly reconstructing words. However, some computational limitations were encountered, an aspect which needs further investigation. The app interface allows musicians to play different songs, both monophonic melodies and polyphonic ones (up to two notes for now). Nevertheless, expressiveness remains quite limited, as it was not possible to implement a proper sound synthesis section. Future work will involve the development of a full synth engine, which will improve timbre customization. Initial tests revealed that the case is easy to use, allowing musicians to easily hold the smartphone without suffering from fatigue. When a final version of the case is finished, an evaluation of this research could be conducted in order to test overall instrument playability. Future work thus involves working on the C++ version of the Faust code in order to introduce optimizations and make the algorithm more efficient, and exploring the instrument's capabilities and limitations with an evaluation and feedback from musicians. The source code of this project can be found at: `github.com/Rickr922/mobile-vocoder`, along with the app in the form of .apk file.

6. REFERENCES

[1] G. Essl and M. Rohs, "Interactivity for mobile music-making," *Organised Sound*, vol. 14, pp. 197 – 207, 08 2009.

[2] R. Michon, J. O. Smith, M. Wright, C. Chafe, J. Granzow, and G. Wang, "Passively Augmenting Mobile Devices Towards Hybrid Musical Instrument Design," in *Proceedings of the International Conference on New Interfaces for Musical Expression*. Copenhagen, Denmark: Zenodo, Jun. 2017, pp. 19–24.

[3] R. Michon, J. O. I. Smith, M. Wright, and C. Chafe, "Augmenting the ipad: the bladeaxe," in *Proceedings of the International Conference on New Interfaces for Musical Expression*. Brisbane, Australia: Zenodo, Jun. 2016, pp. 247–252.

[4] D. Overholt, "The musical interface technology design space," *Organised Sound*, vol. 14, no. 2, p. 217–226, 2009.

[5] E. Spence, "Apple abandons magical 3d touch from the iphone," *Forbes*, vol. Online, 2019, accessed: 2021-09-06. [Online]. Available: www.forbes.com/sites/ewanspence/2019/06/05/apple-iphone-new-magical-feature-embarrassing-deleted-abandoned/?sh=5a1c416439f7

[6] J. Wang, N. d'Alessandro, A. Pon, and S. Fels, "PENny: An Extremely Low-Cost Pressure-Sensitive Stylus for Existing Capacitive Touchscreens," in *Proceedings of the International Conference on New Interfaces for Musical Expression*. Daejeon, Republic of Korea: Zenodo, Jun 2013, pp. 466–468.

[7] G. Geiger, "Using the Touch Screen as a Controller for Portable Computer Music Instruments," in *Proceedings of the International Conference on New Interfaces for Musical Expression*. Paris, France: Zenodo, Jun. 2006, pp. 61–64.

[8] S. O'Modhrain, "A framework for the evaluation of digital musical instruments," *Computer Music Journal*, vol. 35, pp. 28–42, 03 2011.

[9] J. Oh, J. Herrera, N. J. Bryan, L. Dahl, and G. Wang, "Evolving the mobile phone orchestra," in *Proceedings of the International Conference on New Interfaces for Musical Expression*. Sydney, Australia: Zenodo, Jun. 2010, pp. 82–87.

[10] H. W. Dudley, "System for the artificial production of vocal or other sounds," Patent US2 194 298A, 1940.

[11] J. L. Flanagan and R. M. Golden, "Phase vocoder," *The Bell System Technical Journal*, vol. 45, no. 9, pp. 1493–1509, 1966.

[12] Y. Orlarey, D. Fober, and S. Letz, "FAUST : an Efficient Functional Approach to DSP Programming," in *New Computational Paradigms for Computer Music*, E. D. France, Ed., 01 2009, pp. 65–96.

[13] P. R. Cook, "Principles for Designing Computer Music Controllers," in *Proceedings of the International Conference on New Interfaces for Musical Expression*. Seattle, WA, USA: Zenodo, Jun. 2001, pp. 3–6.

[14] G. Wang, "Ocarina: Designing the iPhone's Magic Flute," *Computer Music Journal*, vol. 38, no. 2, pp. 8–21, 06 2014.

[15] R. Michon, C. Dumitrascu, S. Letz, Y. Orlarey, and D. Fober, "The Gramophone: A Programmable DMI to Facilitate the Teaching of STEM Disciplines," in *Proceedings of the 18th Sound and Music Computing Conference (SMC 2021)*. Zenodo, Jun. 2021, pp. 359–364.