A Faust-built Mobile Vocoder Instrument

Riccardo Russo
Aalborg University - Copenhagen
rrusso19@student.aau.dk

ABSTRACT

The growth of increasingly powerful smartphones and their ubiquity opens up more and more possibilities in the creation of NIMEs. However, since these mobile devices 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 opportunities offered by smartphones in creating a portable version of such an old musical effect. Both an app and a phone case were developed. The vocoder app behaves well, showing a clear reconstruction of the words and a quite pleasant timbre. However, some difficulties were encountered in the development process, in particular, some smartphones seem to be not powerful enough to handle a high grade 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 has been traditionally performed by laptops. In particular, for what concerns the audio field, many DSP algorithms, which require high computational power, can now run on relatively cheap smartphones without issues, making these devices suitable for the development of more and more complex NIMEs. While a core aspect, computational power is not the only reason for this fact. Smartphones nowadays include a plethora of sensors for interfacing with the outside world: this opens up many mapping possibilities. Moreover, the ubiquity of these devices provides a wide user base.

The characteristics above mentioned are appealing, however, many areas of limitation need also to be considered when developing a NIME on mobile phones, which can be summarised by the phrase: "smartphones are not conceived for musical purposes". This statement reflects itself in both hardware and software limitations. From the hardware side, the main limitation lies in the touchscreen being the primary for of input: while easy to use, it provides almost no tactile feedback response, a crucial factor in the development of a musical interfaces [1]. Moreover, this type of interaction usually does not capture pressure information; Only Apple did some effort in this sense, by developing the 3D Touch technology. However, the latter was soon abandoned, proving that more sophisticated interfaces are not of commercial interest. Smartphone sensors such as accelerometers and gyroscopes can work as input devices, nevertheless, it has to be noticed that these are not designed for accurate musical mapping, therefore their data need to undergo some pre processing before being used. While the latest smartphones offer quality ADCs and DACs, external input/output interfaces are usually limited to jack and Bluetooth connections, a part from the built-in speakers and microphones. From the software side, the two major OSs, iOS and Android, present different limitations. The first one has always been capable of great audio performance and vast development support, however, prototyping and development on Apple machines usually requires permission from Apple, fact that might discourage amateur developers. Android, on the other hand, has traditionally put less importance on audio processing, focusing on other tasks, nevertheless, it allows for an easy development process with no restrictions.

1.1 Smartphone as a Portable Instrument

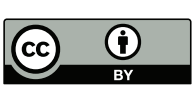
Despite the aforementioned limitations, the high number of opportunities provided by mobile devices are appealing, and several works investigated the opportunities provided by smartphones for the development of NIMEs, sometimes trying to expand the mobile devices functionalities. [2].

An early work by Geiger [3] studied the possibilities offered by the touchscreen itself as a controller. At first, he 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 easy portable and usable in different social contexts
- have an interface which maximises control and give immediate feedback, providing control not only on the note and volume, but also on the sound processing level
- be learn-able and master-able, giving the player direct feedback on his progress

These points are also in accordance with the evaluation principles later provided by O'Modhrain [4].

Stanford Mobile Phone Orchestra (MoPhO) [5] not only showed that smartphone instruments are suitable for social and collective playing environment such as an orchestra, but it proved that these instruments can be interesting means of music production even when expanded with external devices. In fact, in its 2010 evolution players were provided



NIME'20, July 21-25, 2020, Royal Birmingham Conservatoire, Birmingham City University, Birmingham, United Kingdom.

with speaker gloves, in order to provide amplification while keeping focus on portability.

With the growth smartphone popularity, many musical apps were developed, bringing different ideas for the touchscreen usage as a music controller. Bebot [6], is a mobile phone synth with an interesting interface, which was specifically designed for touchscreen devices. Therefore, inside Bebot, virtual knobs, sliders, and even keys leave space for controllers which react to the fingers movements and amount of fingers present on the keyboard. More recently, Electrospit [7] 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, but also provides a companion app. These facts stress how much importance portability has in new musical interfaces design: in fact not only Electrospit transformed a fairly old effect like the talk-box in 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 for being played.

In this work, an Android app version of a Vocoder is developed by using the Faust programming language. The goal is to test the portability opportunities provided by another "classic" musical effect, which usually comes in the form of keyboard synths. The app was developed following 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, the Vocoder was expanded by designing a case which could 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 possile evaluation protocol, and Section 5 concludes the paper.

2. THE VOCODER

The vocoder, invented in 1939 by Dudley [8], 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 previous one. The gain of these filters is set using the coefficients from the first filter-bank, this way the frequency content of the carrier is modulated and becomes close to the original signal. The vocoder was an early attempt at compressing speech information, in fact it only requires to store the filters coefficients. 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, vocoder was extended by Flanagan, who created the Phase vocoder [9]. This version employs the Fourier transform to split and analyse the modulator signal, thus achieving more detail and allowing to keep the phase information. The schematic of a classic vocoder is illustrated in Fig. 1.

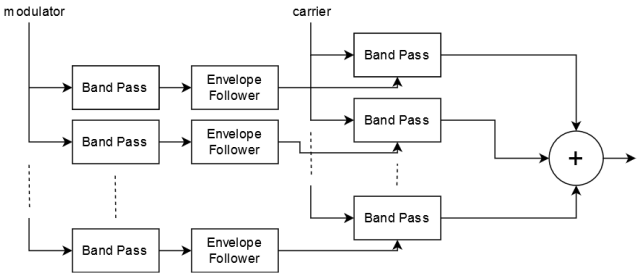


Figure 1: Schematic of a standard vocoder.

3. IMPLEMENTATION

3.1 DSP Algorithm

The vocoder has been implemented using Faust: a high-level, domain specific functional programming language for developing digital signal processing algorithms, which can be exported and integrated into different software. 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. Once developed, code can be automatically compiled to build an Android apk file, this can be downloaded from the Faust database by using a QR code, thus allowing to skip the usual development process. These features were the reason why Faust was chosen for this project.

Given its nature, Faust does not allow to perform FFTs, therefore the classic version of the Vocoder was implemented in this work. Figure 2 illustrates the structure of the algorithm.

The modulator is obtained by speaking into a microphone, then its signal is split: one branch is analysed by a filter-bank made of 74 resonant band-pass filters: this number was chosen as it is a tradeoff between analysis capabilities and requested computational power. The other branch is multiplied by a gain externally controlled and analysed by and envelope follower. The employed filters and envelope follower are part of the integrated 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 the modulator signal is null. This way the gain of the filters is "opened" only when needed and by changing the modulator gain it is possible to control the "sensibility" of the filter bank. 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 outputted from each band of the filter-bank is analyzed by an envelope follower with the same characteristics of the previous 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 was previously scaled by a gain externally controlled, this provides an overall volume control for the vocoder. The carrier signal is made of a blend of a square wave and a sawtooth wave, blend which can be controlled by a slider in order to play either the first one, the second one, or a mixture of those. At first, only the sawooth was chosen, in fact, it contains both even and odd harmonics of the fundamental, allowing for a more accurate reconstruction of the voice; later also the square wave was added, in spite of it containing only the odd harmonics, in order to increase expressiveness.

A Mix value 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 classic synth. On the contrary, when the signal is completely wet, only the vocoder signal is present, meaning that playing a key when not 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

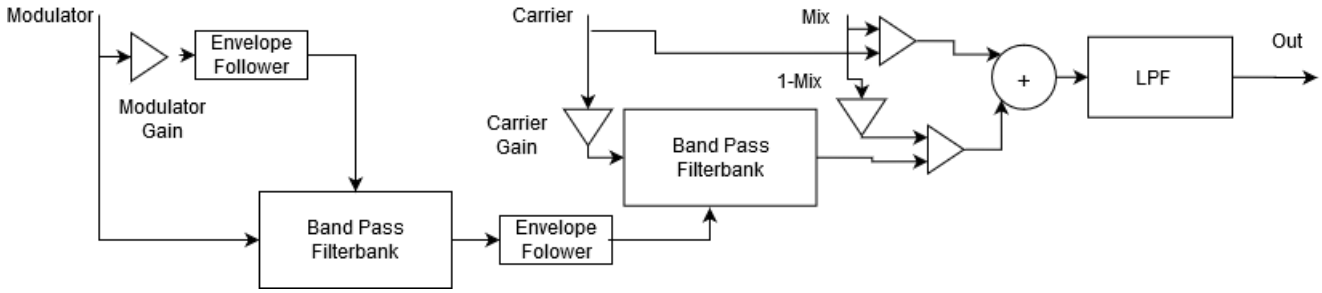


Figure 2: Schematic of the algorithm.

is a 3-pole Butterworth LPF, already present in the Faust library.

The overall sound is quite pleasing, and the algorithm responds without lags, in addition, voice synthesis works well and spoken works are recognisable. Timbre is not very customisable, with the controls being limited to the choice of a sawtooth wave or a square wave, and the low pass filter cutoff frequency. Nevertheless, the focus of this work is more on the vocoder mechanics rather than on developing a full synthesiser. In addition, as it will be seen later, Faust interface does not allow to include many controls, therfore a choice had to be made. Future work could include improvements on the synthesis part of the app. On some smartphones it has been 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 on all the computation. This issue might be solved by optimizing Faust code at C++ level, however further investigation needs to be done in this sense.

3.2 Interface - Mapping

In order to develop the interface, the Faust Smartkeyboard UI [10] tool was used. This 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 the data of the smartphone accelerometers and gyroscopes. However, its easiness of implementation brings also some disadvantages over traditional GUI development tools: for instance, the aspect of graphic elements of the UI cannot be changed from the default one, as such, GUI customization is somehow limited.

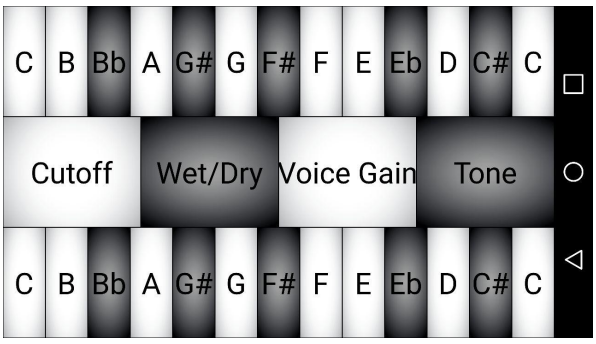


Figure 3: Application interface.

Two traditional looking keyboards control the carrier frequency, this decision was made, instead of developing something more "touch friendly" like what Bebot uses, for two reasons. The first one is that Smartkeyboard is optimised for developing a keyboard-like interface, and changing this is not very straightforward. The second one follows Cook's statement [11]: "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 is valid also for 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 block of the algorithm could potentially provide more parameters to control the instrument, thus increasing expressiveness, however, two main issues arose. The first one is pretty straightforward: the goal was to create an app that could run on smartphones, which have small screens; therefore, adding all the possible controls would have made the interface elements so tiny that playing would have been virtually impossible. The second one is more related to the playability of the instrument: when designing a NIME it is necessary to take a decision on the complexity of mapping, which reflects on the cognitive load on the player [1]. Some low-level parameters might need to be mapped in a way that, after being set, the player can "forget" about them, in order to focus on other aspects of playing. Both issues could have been solved by creating sub-menus in which to place all the "secondary" controls. However, Smartkeyboard does not allow to do this, therefore some parameters had to be set inside the code. In particular the hardcoded parameters were: the attack and release time of the envelope followers (previously mentioned), the number of bands (74) and the Q 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 Fig. 3, the app is intended to be played with the Android keys facing the body, this since the microphone is located at the bottom of the phone, this way, the way of playing resembles the one of flutes. 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 can span two octaves. Both keyboards span one octave, the left keyboard lower 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 control the carrier gain and the overall app volume: 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 filter: the nearer to the bottom of the phone, the higher the cutoff frequency. Obviously, "Dry/Wet" controls the dry/wet

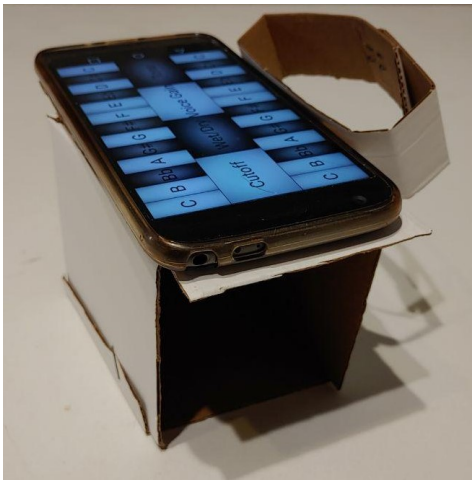


Figure 4: Case prototype.

Figure 5: Case prototype.

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 means full square wave, far from the bottom 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 stopped. This problem particularly affects the present app, since in most smartphones the speaker is located near to the microphone. For this reason, either an external speaker or a microphone are needed while playing the instrument. However, this might mean that the instrument alone cannot follow the first one of the Geiger's principles mentioned in section 1. Moreover, it was noticed that playing with two hands simultaneously is not very comfortable, as one of them is busy holding the phone near to the mouth.

In order to overcome both issues, a case was designed, which is 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 on a tiny rail placed on the handle, this way the case can be adapted for various hand sizes. The bottom box would hold a detachable speaker, in order to provide amplification, and so that projection of sound can be controlled. Moreover, as mentioned by Oh [5], keeping the sound source close to the instrument (the keyboard in this case) can provide a closer association between the instrument and the performer. The speaker box would be able to rotate, in order to better orientate the outcoming sound. The instrument can be played either with the phone microphone or with an external one without affecting the playability.

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 [4], 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, however, this could be further investigated. The manufacturer would only need to produce the case, which is thought to be easily 3-D printed. In fact, users would be able to download the app on their smartphones. This is one of the advantages of developing NIMEs on mobile devices: players already possess the hardware in their pocket. Nevertheless, it would be interesting to investigate the difficulties and the resources needed for the realisation of the case. The most important point of view in the evaluation of this instrument is undoubtedly the performer's one. The aim of this work was to recreate an already existing instrument in a more compact, cheap and portable form without dramatically affect playability. For this reason, it would be important that the player could achieve a decent level of mastery. A straightforward way to test this 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 a nice and pleasant sound, however, some computational power issues were encountered. This aspect needs further investigation. The app interface allows for playing different songs, both monophonic melodies and polyphonic ones (up to two notes for now), nevertheless, expressiveness remains quite poor, as the interface does not allow to create a proper sound synthesis section, which would have increased timbre quality by far. Tests revealed that the case is easy to use and allows to easily hold the smartphones without suffering from fatigue. When a plastic version of the case will be available, an evaluation research could be conducted in order to test the instrument playability. Future work could involve working on the C++ version of the Faust code in order to add controls and increase expressiveness. For instance, it would be possible to add more sound controls, creating a proper synthesis engine.

6. REFERENCES

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

[2] J. Wang, N. D'Alessandro, A. Pon, and S. S. Fels, "Penny: An extremely low-cost pressure-sensitive stylus for existing capacitive touchscreens," in *NIME*, 2013.

[3] G. Geiger, "Using the touch screen as a controller for portable computer music instruments," in *NIME*, 2006.

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

[5] J. Oh, J. Herrera, N. Bryan, L. Dahl, and G. Wang, "Evolving the mobile phone orchestra," in *NIME*, 01 2010.

[6] "Bebot website." `www.normalware.com/`. Accessed: 2020-05-29.

[7] "Electrospit website." `www.electrospit.com/`. Accessed: 2020-05-29.

[8] H. W. Dudley, "System for the artificial production of vocal or other sounds," 1940.

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

[10] "Smartkeyboard website." `ccrma.stanford.edu/~rmichon/smartKeyboard/`. Accessed: 2020-05-29.

[11] P. Cook, "Principles for designing computer music controllers," in *NIME*, 04 2001.