USING THE BUILT-IN MICROPHONE TO ADD VELOCITY SENSITIVITY TO A LAPTOP KEYBOARD

Perttu Hämäläinen, Aki Kanerva, Teemu Mäki-Patola

Helsinki University of Technology
Telecommunications Software and Multimedia Laboratory
{pjhamala,akanerva,tmakipat}@tml.hut.fi

ABSTRACT

In this paper, we describe software that transforms the keyboard of a laptop computer into a velocity sensitive musical keyboard. There is already a variety of "virtual midi keyboard" software available, but PC keyboard hardware does not normally provide information about the dynamics of the keystrokes. Our system remedies this through real-time analysis of the sounds of the keys via the built-in microphone of the laptop. We describe a prototype system, consisting of audio input analysis and synthesis software, and a user interface for adjusting the analysis parameters. Experiences and observations from testing the prototype are also discussed.

A demonstration video of the keyboard can be downloaded from *http://www.perttu.info/dvk/*

1. INTRODUCTION

musical workstations for mobile musicians, thanks to increasing signal processing power and decreasing prices. As Van Veen puts it, "The role of the laptop in artistic production has become ubiquitous: it records, transmits, receives, creates, edits, effects, and performs; it is mobile, fast, and light."[1]. However, laptops usually lack expressive interfaces for real-time musical control, which may make it difficult to harness one's inspiration when there are no midi keyboards or other control devices at hand, e.g., during a train trip or at work.

Laptops natively have a keyboard, a touchpad or joystick, and a built-in microphone as the input devices. The keyboard and touchpad can be used for musical input with software that converts keystrokes and mouse movements into midi messages [2][3]. Sounds can be recorded using the microphone, and sound input features such as amplitude and pitch can be converted into notes or control signals [4][5]. The native input devices can be augmented with lightweight portable midi keyboards (e.g., M-AUDIO Oxygen [6]), recording/playback interfaces, and various sensors (e.g., Phidgets [7]).

This paper describes how a laptop keyboard can be made velocity sensitive by analyzing the sounds of the keystrokes via the laptop's built-in microphone. With the goal of expanding the expressive capabilities of a laptop without additional hardware, we have built a velocity sensitive keyboard software prototype, shown in Figure 1. Basically, the microphone is treated similar to piezoelectric sensors in MIDI trigger devices such as drum modules. The dynamics information provided by the microphone is augmented with hit location provided by the keyboard

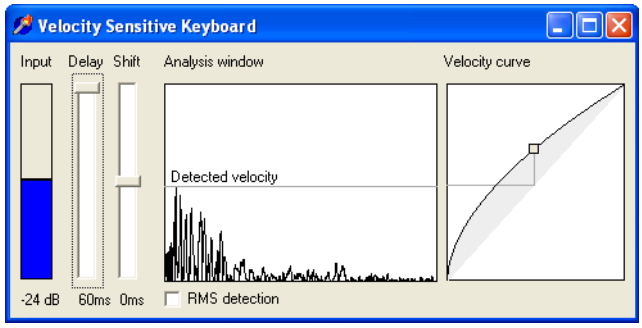

Figure 1: *A screenshot of the software.*

2. SYSTEM DESCRIPTION

The prototype continuously records incoming audio into a ring buffer. Keyboard events are delayed for a user-specified time, and when processing a key event, the audio recorded during the delay is analyzed. The results of this analysis are used to control sample-based audio synthesis.

2.1. Software and Hardware Environment

The prototype runs on a Windows XP laptop with a VIA Vinyl AC'97 WDM audio driver. To minimize latency, an ASIO4ALL driver is used on top of the VIA driver. ASIO4ALL is a hardware-independent low latency ASIO driver that can be used even with chipsets with no real ASIO drivers [8]. Audio input is read and output written in an audio I/O callback generated by the PortAudio API [9] with 44,100 samples per second and a buffer size of 64 samples (the minimum supported by the driver). Keyboard is accessed using the Microsoft DirectInput API [10].

2.2. Audio Analysis

Figure 2 shows an example of a keystroke sound recorded via the built-in microphone of a laptop. In the test system, the length of a recorded keystroke is approximately 150ms. This corresponds to a sixteenth note at 100 beats per minute. The first step in the analysis is to shorten the sounds by high-pass filtering, which removes the slowly decaying low-frequency resonances of the laptop body and the underlying surface. This ensures that the sounds of consecutive keys don't overlap in the analysis. We use a second-order IIR Butterworth high-pass filter with 600Hz cutoff frequency. An

example of a filtered keystroke is shown in the lower part of Figure 2. The filtering also removes the low-frequency hum that is common in laptop microphones.

After the filtering, the maximum absolute value of the input signal is found within the analysis window and mapped to the gain of the played samples. Alternatively, the RMS (square root of the mean of squared signal values) of the window can also be used. The maximum value is easy to visualize, as shown in Figure 1. On the other hand, RMS is less sensitive to noise because it is an averaged feature.

Note that the term 'velocity sensitive' is used here only to adhere to the common terminology of musical keyboards and MIDI. A MIDI note-on message contains a velocity parameter [11], but we are not measuring the true velocity of a key. The sound of a keystroke is proportional to kinetic energy absorbed from the key, which is a function of velocity, but also a function of unknowns such as the physical coupling and masses of the key and the player's body parts. The validity of velocity as a control parameter depends on the synthesis method. For physically based synthesis, analyzing key sounds can actually be considered more realistic, as real vibration gets mapped to simulated vibration.

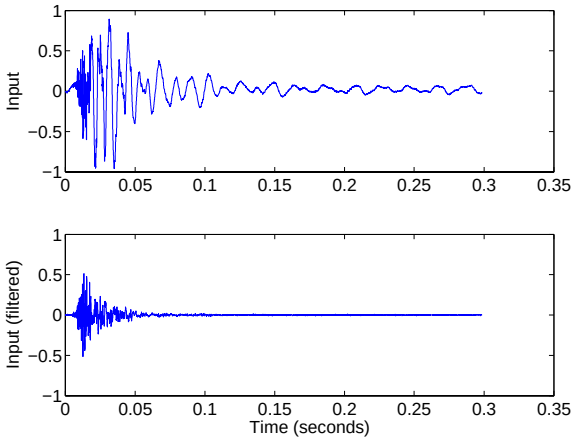

Figure 2. Raw and high-pass filtered keystroke audio.

2.3. Latency

For real-time playing, it is crucial to minimize the latency of the system, that is, the delay between a keystroke and the start of a synthesized note. Latencies of less than 10ms are often suggested for musical controllers [12][13]. Finney has shown that delay in auditory response can cause large errors in the performance of pianists [14]. However, Sawchuck *et al.* suggest that latency tolerance is dependent on the piece of music and instrumentation [15].

Latency is mainly caused by the following three components:

1. The size of the audio analysis window. There's a tradeoff between latency and the robustness of the analysis results. Shorter window sizes yield lower latency, but more random dynamics information. In an informal test with five users who played the prototype in maximum value detection mode, the preferred analysis window sizes ranged from 10 to 35ms, with median 12ms. The optimal value depends on each player's tolerance for latency and imprecision of dynamics. A player with ten years of drum playing experience commented the latency was not annoying with window sizes less than 10ms, but the dynamics were better above 10ms.
2. Audio hardware and drivers. MacMillan et al. report audio input-output latencies as high as 60-120ms in the Windows operating system with DirectSound drivers of consumer audio hardware [16]. In the test system, the ASIO4ALL driver reports less than 1 ms output latency and 4.4 ms input latency. This was confirmed by our measurements.
3. Keyboard hardware and drivers, that is, the delay between the actual key press and the notification of a key event in the software. Unfortunately, DirectInput does not provide information about keyboard latency. In the test system, a key event arrives within 1ms of the sound. Combined with the 4.4 ms audio input latency, this indicates approximately 5 ms of keyboard latency.

In addition to the components above, there are random small latencies related to operating system thread scheduling. Neglecting these, the total latency becomes $\max[L_k,(L_{ai}+L_w)]+L_{ao}$, where L_k is keyboard latency, L_{ai} is audio input latency, L_w is the length of the analysis window and L_{ao} is the audio output latency. Only the larger one of L_k and $L_{ai}+L_w$ has an effect – information about key location and dynamics can arrive asynchronously, but both are needed for generating a note-on message for the synthesis. The analysis window can be shifted in relation to keyboard events so that the sound of a key starts at the beginning of the window.

2.4. User Interface

The user interface of the prototype has sliders for adjusting the analysis window size (key event delay) and shift. Additionally, an input level meter helps in adjusting microphone input gain.

To visualize the analysis, the user interface shows the contents of the analysis window, that is, rectified input recorded after the last keystroke. The window shows how much of the sound fits inside the window and what part of the waveform gets analyzed. The horizontal grey line shows the detected velocity. An adjustable velocity curve allows fine-tuning the dynamics. The line from the waveform display continues to the velocity curve so that the user sees its effect on the detected velocity.

The velocity curve is essential, since the dynamic behavior of a laptop keyboard is quite nonlinear. The sound typically consists of several parts: the soft sound of a finger hitting the key, the spring mechanism, and the sound of the key colliding against the laptop when fully depressed. The collision is a particularly nonlinear component, as it becomes audible only when a key is pressed hard enough to reach the limits of its movement range.

3. DISCUSSION

At the time of writing this, the prototype has been tested by seven people with several years of musical training. All of them were very enthusiastic about the software, although it can take time adjusting to play on a laptop keyboard. The keys are small so that it is easy to accidentally hit two keys at the same time. To maximize the accuracy of the detected velocity, you should also avoid hitting the body of the laptop with your palms when playing.

The prototype has the same limitations as 'real' velocity sensitive musical keyboards. Keyboards are best for controlling instru-

ments, where notes are isolated events that are not interacted with once triggered. Examples of this are piano and many percussion instruments, at least if only considering a subset of playing styles. A keyboard is not an ideal control device for continuously controlled instruments, such as woodwinds and bowed strings (see, e.g., Paradiso and O'Modhrain [17]).

Another limitation of laptop keyboards is that they are not built to handle several simultaneous key presses. Keys can block each other, which makes the keyboard best for playing melodies and percussion instead of chords. For chord support, the audio analysis should also be improved, as it cannot currently determine the individual velocities of simultaneously pressed keys.

We used a sampled djembe drum in our tests, which worked well. Several samples were recorded by hitting the drum at varying locations and varying intensities. In the synthesis, key location was mapped to hit location. In this aspect, a qwerty keyboard has more freedom than a piano keyboard. We mapped each row of keys to different distance from the center of the drum membrane.

4. CONCLUSIONS AND FUTURE WORK

We have presented a prototype of velocity sensitive keyboard software that combines information from a regular keyboard and a microphone. A demonstration video can be downloaded from *http://www.tml.tkk.fi/~pjhamala/keyboard/*. The software has been received with enthusiasm by end users, although there's a tradeoff to be made between latency and accuracy of dynamics. The software is not ideal for live performance due to interfering sounds, but it appears promising for headphone use in a reasonably quiet environment.

We aim to integrate the software with other musical tools, e.g., as a PD external. In software/hardware environments with high latency, the software can be used with non-real-time musical editors such as step recorders. There may be other applications to explore too, such as measuring the emotions of the user. A computer game study by Brown and Sykes suggests that the pressure used to press the buttons on a gamepad varies as a function of game difficulty [18].

The user interface is currently quite technical and probably not very intuitive for the less audio savvy users. The interface could be simplified by determining the correct analysis window shift automatically, possibly via onset detection of keystroke sounds. However, adjusting the shift changes the feel of the keyboard, and in preliminary tests, automatic changes while playing felt awkward. This indicates that manual control should be at least an option, and more user studies are needed to make the interface more intuitive.

5. ACKNOWLEDGEMENTS

The work has been partly funded by Academy of Finland. Hämäläinen came up with the original idea and developed the software based on an initial PD-prototype implemented by Kanerva. This report was written by Hämäläinen and Mäki-Patola.

6. REFERENCES

[1] Van Veen, T., Laptops & Loops: The Advent of New Forms of Experimentation and the Question of Technology in Experimental Music and Performance. Presented at UAAC 2002, University of Calgary, Alberta.

[2] Granucon Virtual MIDI Keyboard, http://www.granucon.com/vmk.html, link visited 26th Jan 2006

[3] VKeys virtual midi keyboard, http://www.tml.tkk.fi/~pjhamala/vkeys/, link visited 26th Jan 2006

[4] Furukawa, K. and Dutilleux, P., Live-electronics Algorithms in the Multimedia Work 'Swim Swan', Proceedings of the 5th Int. Conference on Digital Audio Effects (DAFx-02), Hamburg, Germany, September 26-28, 2002

[5] Igarashi, T. and Hughes, J.F., Voice as Sound: using non-verbal voice input for interactive control, Proceedings of Symposium on User Interface Software and Technology (UIST'01), ACM Press, 2001

[6] M-AUDIO Oxygen 8 keyboard, http://m-audio.com/products/en_us/Oxygen8-main.html, link visited 25th Jan 2006

[7] Phidgets Inc., http://www.phidgets.com/, link visited 26th Jan 2006

[8] ASIO4All Universal ASIO Driver, http://www.asio4all.com/, link visited 26th Jan 2006.

[9] Bencina, R., Burk, P., PortAudio – an Open Source Cross-Platform Audio API, Proceedings of International Computer Music Conference (ICMC'01), 2001, available online at http://www.portaudio.com/docs/, link visited 26th Jan 2006

[10] Microsoft DirectInput Overview, http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dninput/html/diov.asp, link visited 26th Jan 2006

[11] Heckroth, J., Tutorial on MIDI and Music Synthesis, http://www.harmony-central.com/MIDI/Doc/tutorial.html, link visited 26th Jan 2006.

[12] Freed, A. Chaudhary, A. Davila, B., Operating Systems Latency Measurement and Analysis for Sound Synthesis and Processing Applications, Proceedings of International Computer Music Conference (ICMC'97), 1997

[13] Wright, J. Brandt, E. System-Level MIDI Performance Testing, Proceedings of the International Computer Music Conference (ICMC 2001), 2001

[14] Finney, S.A., Auditory Feedback and Musical Keyboard Performance. *Music Perception*, vol. 15, no. 2, pp 153-174, 1997

[15] Sawchuk, A. A. Chew, E. Zimmermann, R. Papadopoulos, C. Kyriakakis, C., From Remote Media Immersion to Distributed Immersive Performance. *Proceedings of 2003 ACM SIGMM workshop on Experiential Telepresence*, 2003

[16] MacMillan, K., Droettboom, M. and Fujinaga, I. Audio Latency Measurements of Desktop Operating Systems, *Proceedings of International Computer Music Conference (ICMC'01)*, 2001

[17] Paradiso, J.A., O'Modhrain, S., Current Trends in Electronic Music Interfaces, *Journal of New Music Research*, Vol. 32, No. 4, 2003

[18] Sykes, J. and Brown, S., Affective gaming: measuring emotion through the gamepad, *CHI '03: CHI '03 extended abstracts on Human factors in computing systems*, pp. 732-733, ACM Press, 2003