Audio Engineering Society

Convention Paper

Presented at the 127th Convention
2009 October 9–12 New York, NY, USA

Editing MIDI Data Based on the Acoustic Result

Sebastian Heise[1], Michael Hlatky[1], and Jörn Loviscach[2]

[1] Hochschule Bremen (University of Applied Sciences), 28199 Bremen, Germany
sebastian@h3e.eu, mhlatky@acm.org

[2] Fachhochschule Bielefeld (University of Applied Sciences), 33602 Bielefeld, Germany
joern.loviscach@fh-bielefeld.de

ABSTRACT

MIDI commands provide an abstract representation of audio in terms of note-on and note-off times, velocity, and controller data. The relationship of these commands to the actual audio signal is dependent on the actual synthesizer patch in use. Thus, it is hard to implement effects such as compression of the dynamic range or time correction based on MIDI commands alone. To improve on this, we have developed software that silently computes a software synthesizer's audio output on each parameter update to support editing of the MIDI data based on the resulting audio data. Precise alignment of sounds to the beat, sample-correct changes in articulation and musically meaningful dynamic compression through velocity data become possible.

1. INTRODUCTION

Even simple editing tasks on MIDI data are difficult due to product-specific algorithms and sound-specific differences. For example, sample-accurate quantization of a beat produced by a drum synthesizer for example is not possible if the duration of the attack phases of the drum samples are not perfectly identical. Compression of the dynamic range is in principle possible based on simple computations on MIDI velocity data. This feature is implemented in most standard MIDI sequencer software. However, the exact velocity value mapping the plug-in's audio output volume is not known, as the velocity may influence other parameters of the plug-in, as for instance the cutoff frequency of a filter. Editing the dynamic range based on MIDI data is therefore hard to control without the implementation of further parameter controls. Another issue with the editing of raw MIDI data is that it is difficult to precisely edit the performance articulation such as for instance to achieve accurate legatos if one does not know the precise decay characteristics of the synthesizer patch at hand.

The MIDI editor described in this paper forms a feedback loop by taking the actual audio signal into account that is produced by a VST standard software synthesizer plug-in. The user can for instance use the peaks in the audio output of a drum sampler for precise beat quantization. Likewise, as also the exact audio decay characteristics for each MIDI event are known to the system, the system can automatically tweak individual MIDI events in the score in order to achieve more natural sounding staccato and legato effects. Furthermore, with the audio data at hand, the exact audio level envelope for each MIDI event can be employed to make informed level adjustments on the basis of MIDI velocity and volume controller data. Thus, it is possible to produce sophisticated compression or expansion control curves of the dynamic range.

2. RELATED WORK

MIDI [1] commands have been used for storage and transfer of musical information since the standard was defined in 1982. MIDI data hereby carries information about the pitch of a tone to be synthesized, as well as its associated volume and duration. As the synthesizer used to generate sound is dependent on the user, it is not possible to predict the resulting sound output based on the MIDI data alone.

Some software, such as Celemony's Melodyne [2] or Steinberg's Cubase VariAudio [3] allow for the extraction of MIDI data from existing audio material. In these editors, audio data is analyzed in terms of pitch over time, and from this, MIDI notes are generated, which can be used as a template to process the audio or can be fed into an additional synthesizer. This could be used to double up a vocal melody with another instrument.

MIDI data has also been used is audio compression applications. Modegi [4], for instance, extracts sinusoidal waveforms from audio source material. The sinusoidal waveforms are approximated by harmonic complex functions over time. This information is coded in MIDI commands, which are transferred from a sender to a receiver. The receiver feeds the MIDI data into a synthesizer instrument, which produces again audio data.

In a previous work [5], we described how to render independent waveform plots for each MIDI event onto a piano-roll style editor. For that purpose, we developed a system, which silently computes the audio output of VST synthesizers. We use a proprietary host software that runs several instances of a VST plug-in in a background thread. The rendering of thousands of waveforms onto the computer screen in real-time is carried out directly on the graphic cards. In that work, we used color to provide feedback about different timbre regions on the piano-roll editor. For every MIDI event, the zero crossing rate (ZCR) of the corresponding audio data is extracted and mapped to the hue value of the HSB color space. The result is a color tint for each, reddish areas thereby representing mellow sounds, and bluish areas representing more bright sounds.

In this work we extend our previous software to provide a feedback loop for advanced automated editing of the MIDI input data. The audio data of the VST synthesizer plug-in is analyzed and, depending on settings provided by the user, changes on the audio data's generating MIDI data can be carried out.

3. IMPLEMENTATION

This work presents two advanced MIDI editing modes in the time domain: Sample-accurate beat quantization and sample-accurate performance articulation such as legatos. Furthermore, three different dynamic level adjustment modes are introduced, based on the note-on velocity data only, on polyphonic aftertouch, and, in the third mode, based on channel volume messages. In each of the five modes, the editing decisions are automated, system-driven and based on the synthesized audio output of the corresponding MIDI events.

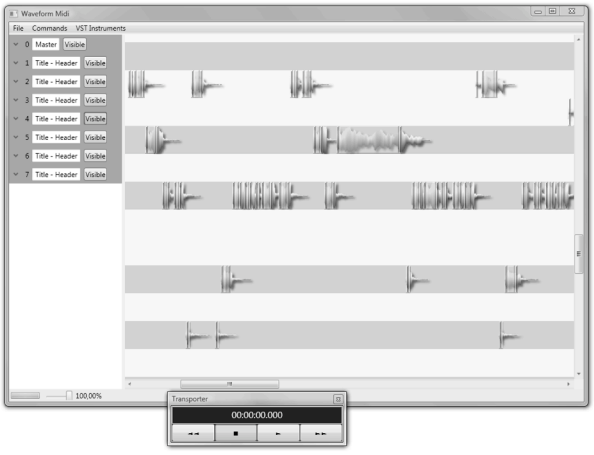

Fig. 1: The user interface of the software. It resembles a common piano-roll style MIDI editor, the menu, however, provides access to the advanced MIDI editing capabilities described in this work. In addition to this, the

corresponding waveform for each MIDI note is plotted on top of the note's bar.

3.1. Sample-Accurate Beat Quantization

Quantization of non-aligned and de-quantization of aligned MIDI events are features often used in MIDI editing: Apple's Logic for instance provides a so-called "humanize" function for the de-quantization of MIDI events, and allows for the quantization of raw MIDI data to different bars. It is common that composers, of electronic music in particular, want to align sound samples precisely to the beat of a musical composition. After the user has entered the MIDI data manually, using for example a MIDI keyboard, the quantization commands can be used to align the recorded MIDI events to a previously defined time grid. In standard MIDI editors, the note-on and note-off events times are therefore aligned with this time grid. This quantization process works well, providing attack characteristics of the different MIDI event's corresponding synthesized audio data are fairly similar. In the case that they are not, the non-precisely aligned sound's attacks cause the beats to fall apart.

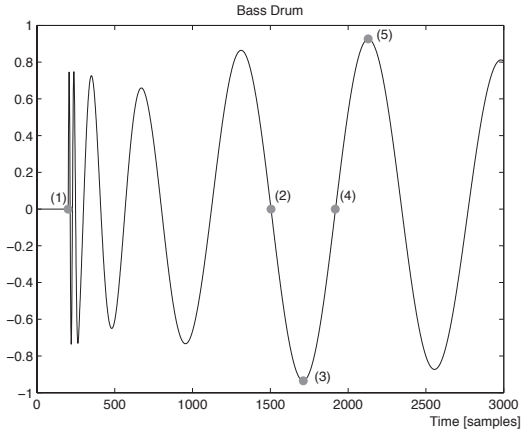

Fig. 2: Waveform plot of the first 3,000 samples of a synthesized bass drum sound, taken from the Freesound database[1]. The five possible quantization markers are displayed as grey bullets. The sound is not edited sample-precisely, and therefore takes exactly 197 samples until the actual attack (1). Furthermore the 1711th sample (3) has the maximum negative value, the corresponding closest-to zero-crossing sample is at 1505th position (2). The maximum positive value sample is at 2130th position (5), the corresponding closest-to zero-crossing sample at 1919th position (4).

Our system can precisely analyze the audio attack characteristics of each MIDI note and readjust the MIDI event's times exactly, depending on three different options: The first option provides an alignment to the audio data's first sample with a value different from zero; the second provides an alignment to the audio data's sample with the greatest positive or negative value; the third option picks the ultimate zero crossing occurring before the sample with the highest positive or negative value as an alignment point. This is supposed to ensure maximum correlation between different waveforms.

In Fig. 2, a typical synthesized bass drum sound is shown. If normal MIDI quantization was applied on the sound, the attack would have a delay of 197 samples. Our software can precisely align the attack of this sample to the time-grid by shifting the MIDI note-on command forward by a time corresponding to 197 audio samples. Furthermore, our software supports an advanced quantization functionality: The sound's alignment position could also be sample 1707, the sample with the maximum negative value, or sample 1505, the closest-to zero-crossing sample before the maximum negative value. Correspondingly, the sound could also be aligned to the positive maximum value sample at position 2130, and the closest-to zero-crossing sample before this at position 1919.

3.2. Sample-Accurate Performance Articulation: Legato

Standard MIDI editing software allows for the processing of MIDI notes to achieve legato effects. However, this only works if the MIDI note's corresponding audio ends with the MIDI note-off message. This is normally not the case in standard audio synthesizers. The audio wave corresponding to a MIDI note can end even before the note-off event, or it may last substantially longer. The automated editing of accurate musical articulation is therefore seldom possible. It is necessary to carry out extensive manual editing to achieve this commonly desired effect. Our software analyzes the audio data corresponding to each MIDI note; therefore it knows the detailed decay characteristics of each sound. For the editing of the musical articulation, a threshold for the decay of the sample on which a new sound shall start can be adjusted by the user, for instance -60 dBFS RMS. The

[1] http://www.freesound.org/samplesViewSingle.php?id=2085

MIDI note-off event of this sound is then delayed, so that the sound lasts longer.

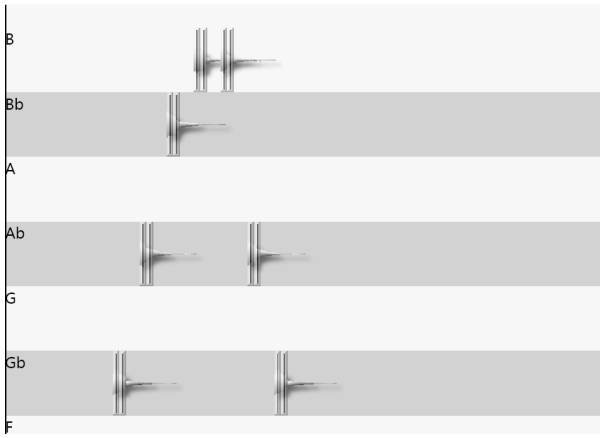

Fig. 3: Midi notes on the piano roll editor with corresponding waveforms plotted on top. In the particular setting for the synthesizer used, the plug-in produces audio although it already received the note-off message. The synthesized audio of the MIDI notes displayed in this figure obviously is not legato.

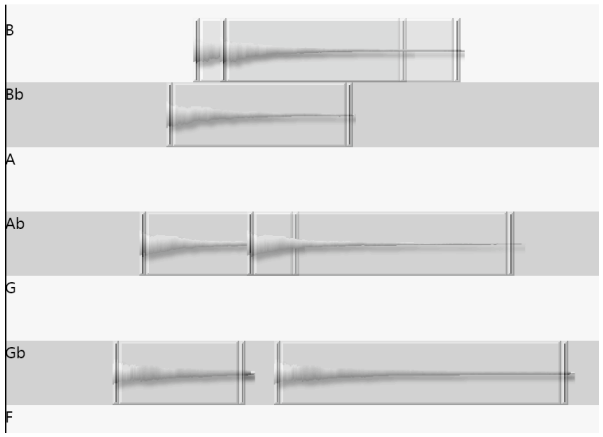

Fig. 4: The same MIDI notes as in Figure 3 after the legato effect is applied. The resulting sound is legato, which can be seen from the waveforms: the audio produced by the synthesizer now overlaps.

3.3. Level Adjustments

Standard MIDI editors allow remapping MIDI dynamics. In particular, every incoming note-on's velocity value may be multiplied with a fixed number. This, however, does not provide precise level adjustments of the synthesizer's audio, as the mapping of the input velocity to the produced audio's levels is unknown. This situation is rectified in the level adjustment mode of our system. For all 127 possible velocity value of every MIDI note of the synthesizer patch at hand the system precomputes the corresponding audio levels.

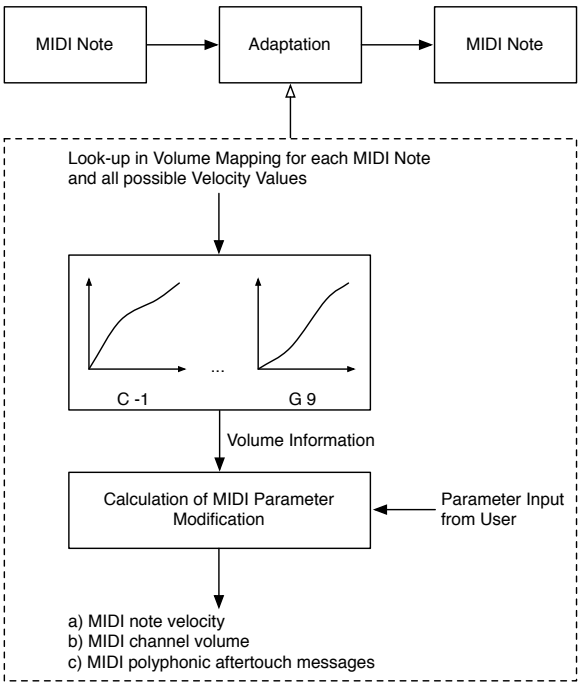

Fig. 5: Schematic overview of the MIDI compression functionality. For each MIDI note, a look-up in a precomputed database of MIDI velocity to volume mapping is performed. Depending on user parameter input of maximum volume, threshold and compression ratio, the volume correction is computed. Furthermore, the user needs to decide whether the MIDI note's velocity or channel volume are affected by the compressor, or if polyphonic aftertouch messages are to be generated by the system to carry out the volume adjustments.

Then one of three level adjustment modes can be picked. In the first mode, a maximum level in dBFS RMS of the produced audio can be defined which shall not be exceeded, and the synthesizer's MIDI note-on messages velocity values are limited correspondingly. The user can choose whether the single MIDI channel volume level or all synthesized audio volume level is used in the maximum level definition. This allows for the processing of a single MIDI channel's, or the overall volume. The effect comes close to a brick-wall limiter.

However, no compression on the actual audio data is carried out.

In the second mode, more sophisticated compression effects become available by allowing the user to chose a threshold in dBFS plus a ratio value. The synthesizer's output of the sequence at hand is analyzed; the MIDI data are marked for portions in which the audio exceeds the set threshold level. During playback of such a portion, the MIDI channel's volume value is attenuated depending on the desired ratio and the audio data output level. This means, a look-up is performed on how much the MIDI channel volume value has to be adjusted to achieve the desired output volume. This enables more sophisticated compression effects.

In the third mode, MIDI polyphonic aftertouch messages are used to achieve a compression effect. This use of aftertouch looks surprising at first, but comes in handy as sample-based piano or drum synthesizer plug-ins often map different audio samples to be played back already depending on the input velocity to achieve a more natural sound. The analysis of the output value is performed as described before.

3.4. Software

The Software has been implemented in C# using the Microsoft .NET 3.5 framework. It provides a graphical user interface similar to the standard piano-roll editor commonly found in digital audio workstation software. The interface also offers familiar interaction features such as drag&drop of MIDI notes. The software is based on a proprietary VST host that allows audio rendering tasks to run in the background. While the user edits the score data or the synthesizer's settings, the waveform data are computed in parallel processing threads. The rendering of a plug-in's audio data of an underlying standard MIDI score takes only fractions of a second on a modern multicore computer: As the synthesized sounds are not listened to, we can run several copies of a software synthesizer at full speed in parallel without artificially throttling them to the playback sampling rate.

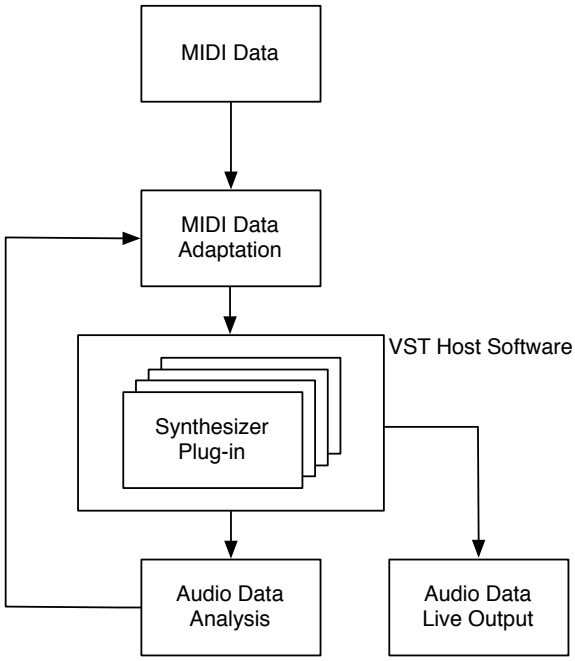

Fig. 6: Schematic diagram of the software's mode of operation. The user provides MIDI data. Furthermore, the user subsequently decides how the MIDI data will be adapted in terms of time correction or volume adjustments. The VST host software runs several instances of the synthesizer plug-in in parallel, allowing live audio output while the audio data is analyzed. On basis of the analyzed audio data, changes are affected on the original MIDI data.

4. CONCLUSION AND OUTLOOK

We have presented a system that can do precise adjustments of MIDI data based on user input and the actual audio output data produced by a synthesizer plug-in. To accomplish this, our system analyzes the synthesizer's audio output data in a background thread and performs adjustments on the MIDI input data, thereby forming a feedback loop.

The analysis is performed without playing back the resulting audio. Therefore, the plug-in's computational speed is not limited to the playback sampling rate.

Future updates of the software may support equalization in the frequency domain, multiband compression, and the attenuation or amplification of single MIDI notes based on their actual audibility in the mix. Furthermore, in future implementations, the characteristics of a synthesizer may be learned by our VST host by means of machine learning and neural networks, thus reducing the amount of required rendering tasks in the pre-computation for the volume correction.

5. REFERENCES

[1] Moog, R.A.: MIDI: Musical Instrument Digital Interface. JAES Volume 34 Issue 5 pp. 394-404, 1986.

[2] Celemony: Melodyne. Retrieved July 13, 2009, http://www.celemony.com/cms/, 2009.

[3] Steinberg: Cubase. Retrieved July 13, 2009, http://www.steinberg.net/en/products/musicproduction/cubase5_product.html, 2009.

[4] Modegi, T.: Very Low Bit-rate Audio Coding Technique Using MIDI Representation. Proceedings of the 11th international workshop on Network and operating systems support for digital audio and video. Port Jefferson, New York, United States. Pages: 167 - 176, 2001.

[5] Heise, S., Loviscach, J.: Waveforms, not Bricks: A Visually Enriched MIDI Editor. Presented at the AudioMostly Conference, (Glasgow, Scotland, September 2-3 2009), 2009.