PRESERVING, RESTORING, AND PASSING DOWN VIDEO-GAME MUSIC FROM THE PAST: THE CASE OF DIRECTMUSIC

Luca A. Ludovico, Alberto Mattea
Laboratory of Music Informatics (LIM)
Department of Computer Science
University of Milan
luca.ludovico@unimi.it

Davide A. Mauro
Department of Computer and
Information Technology
Marshall University
maurod@marshall.edu

ABSTRACT

Video-game music is a form of art that is gaining increasing interest, not only in the technological field but also in musicological research and in the area of Digital Humanities. Characterized by vanishing technological support and by the discontinuing of the software and hardware once used for its reproduction, the establishment of archives and standards that allow the fruition of such artifacts become of crucial importance. The goal of this project is to preserve, restore and pass down video-game music currently available only in an obsolete format. Our main case study will revolve around Microsoft *DirectMusic* technology, adopted in the 90s for composing music for video games and later discontinued. This work can be seen as a first effort at a larger goal that includes a discussion related to the philological process of *what* needs to be preserved and *how*, and the creation of accessible archives.

1. INTRODUCTION

Unfortunately, unlike other forms of musical expression, video-game music is often linked to digital products whose life cycle comes to an end. Not only are commercial products such as video games withdrawn from the market, but the technologies necessary for their execution (hardware architectures, operating systems, etc.) soon become obsolete, preventing the experience of the heritage of musical pieces from the past.

In this sense, the case of *DirectMusic* is particularly relevant. First released by Microsoft in 1996, it became a component of the Microsoft DirectX API. Its goal was to support music and sound effects and provide flexible interactive control over the way they are played. In 1999, *DirectMusic* was introduced as part of version 6.1 of the DirectX library and included in Microsoft Windows operating systems. The first operating system embedding *DirectMusic* was Windows 98 Second Edition. A few years later, *DirectMusic* was deprecated (e.g., it was not available to Windows Vista 64-bit applications), thus making many video-game soundtracks unavailable to new users.

In our opinion, the artistic quality, technical characteristics, and historical considerations make some musical artworks worthy of being preserved. The goal of the project is to preserve, restore and pass down video-game music currently available only in an obsolete format. Our main case study will revolve around the *DirectMusic* technology by Microsoft. Adopted in the 90s and used for composing music for video games, it was later discontinued, making it impossible to access such artworks.

This paper is structured as follows: in Section 2 we will address the importance of preserving video-game music, in Section 3 we will analyze the specifications of the *DirectMusic* file format, in Section 4 we will provide details about the design and implementation of a suitable software tool to convert *DirectMusic* soundtracks into currently in-use standard formats (e.g., MIDI), and, finally, in Section 5 we will draw conclusions and list some directions for future work.

2. PRESERVING VIDEO-GAME MUSIC

The first question to answer is *what* do we want to preserve of the original composition.

Dealing with classical music, for example, would be quite straightforward: usually, there is a reference score, namely a list of music symbols, that provides a logical representation of any expected performance. Needless to say, the audio rendering of a given score can greatly vary from performance to performance, depending on the interpretation and the technical skills of musicians, nevertheless, the result is somehow predictable. Even in this well-established field, debates among experts may emerge, e.g., when a piece presents multiple versions, or performance practices diverge from written music. Other genres and composition techniques, for example, jazz or aleatoric music, are less prone to an unambiguous interpretation of a reference score. In these cases, preserving the original composition, or even choosing a paradigmatic performance, can become a hard task.

Non-linear media and dynamic environments such as those of video games clearly emphasize this problem [1]. When the music is rendered in real-time thru the algorithmic manipulation of some building blocks, completely different renderings of the same soundtrack are possible, often depending on the in-game performances of the player. A large number of different music and sound elements can coexist within a unique mixture, and the piece du-

ration itself becomes an unknown parameter. For example, sound samples can be triggered unaltered at specific points in time (e.g., "Game Over" sound) while a non-deterministic engine combines and manipulates small excerpts or whole music sections, originating from symbolic formats, computer-driven performances, or sound samples.

Going back to the example of traditional music, there are basically two ways to identify music information: i) as it regards symbolic content, gaining access to the score; ii) as it concerns audio content, referring to available recordings. Please note that score availability would allow not only the preservation of symbolic content, but also the production of new audio content; and, on the other side, the availability of audio performances in absence of the original score would give the possibility to reconstruct symbolic information, at least to some extent. An example is provided by the transcriptions of jazz improvisations.

Once again, the scenario of video-game music is far more complex. In fact, in most cases, there is no fixed score and no reference performance. While a rendering can be generated and captured during the game-play phase (assuming it is still technically possible to run such a game), this might not be enough to capture the richness of the original composition. Yet, for many video games, gaining access to the source code and the engine used for their creation is impossible, so recordings remain the only viable solution when the entire artifact cannot be preserved.

Now, after *what* should be taken into consideration, a second research question is emerging: *how* can we preserve it?

The subject of digital preservation, intended here in general terms, has been addressed in a great number of scientific works. For example, in a paper dating back to 2001, Chen discussed the so-called paradox of digital preservation: digital information is expected to be maintained intact, but it is accessed in a dynamic use context and, unfortunately, it is plagued by short media life, obsolete hardware and software, slow read times of old media, and defunct documentation and technologies [2]. In 2006, Gladney enunciates the principles for digital preservation, defined as "the mitigation of the deleterious effects of technology obsolescence, media degradation, and fading human memory", addressing prominent epistemological issues, communication-related problems, and big challenges (e.g., persuading information providers to write metadata) [3]. Another relevant work in this field was authored in 2018 by Owens, who presented the theory and craft of digital preservation starting from the traditions and considering the nature of digital objects and media [4].

As mentioned before, providing a comprehensive review of the scientific literature would be out of scope. In the rest of the paper, we will mainly focus on the format chosen to preserve video-game music. The characteristics that such a format should be present are: being standard and currently in use, being well documented and possibly open, presenting no patent or royalties, and having a long-time future perspective. In addition, we will also address the goal of passing down preserved music works by implementing a dedicated web platform.

The choice of the most suitable format can be also driven by the characteristics of the domain to be described. In our case, the context is video-game music whose score is not available and, in general, not uniquely defined, and whose audio rendering may be greatly influenced by the hardware in use and the player's on-the-fly performance.

A possible approach is analyzing the problem from multiple points of view and trying to preserve as much information as possible. In this sense, formats aiming at a comprehensive description of music, such as IEEE 1599, MEI, and MusicXML/MNX, can be employed [5]. Focusing on IEEE 1599, this XML-based standard in its first version was characterized by a 6-layer structure able to accommodate general, logic, structural, notational, performance, and audio information. In the context of this work, this implies the possibility to encode metadata (general layer), symbolic aspects (logic layer), notated music (notational layer), computer-driven performances (performance layer), audio excerpts (audio layer), and even relationships between music objects (structural layer) within a single document. Moreover, IEEE 1599 can be profitably used as an interchange format, too [6].

Another approach when the original asset is not present at all or not available in a standard and in-use format consists in designing an ad-hoc process to acquire as much information as possible and encode it in a current, easily accessible representation. In this paper, we will follow the latter approach and employ the Standard MIDI Files (SMF) format to this goal. The aforementioned challenges are compounded with issues related to copyrights. It is not always clear, looking at software from ten, twenty, or thirty years ago, what happened to the copyright holders and how to acquire those rights.

On one hand, we have game developers and publishers who use licensed music in their products, and, on the other, we have composers and sound designers that are hired for specific titles, and those compositions exist only for the video game.

While it is true that in recent times more and more videogames soundtracks are released independently from the original game this does not cover the entire industry.

3. THE DIRECTMUSIC FORMAT

3.1 History

In 1995 Microsoft introduced DirectX, a set of application programming interfaces (APIs) with the aim of handling tasks related to multimedia, especially game programming and video. DirectX allowed developers of games and other interactive content to access specialized hardware features without having to write hardware-specific code [7]. Among this collection of APIs, the *DirectSound* one was intended to provide a low-latency interface to sound card drivers. Since it was essentially a low-level interface, it soon became clear that a higher level of abstraction was needed to simplify and standardize the process of sound and music creation.

To this end, in 1996 the *DirectMusic* API was introduced, initially released as an ActiveX control called Interactive

Music Architecture (IMA) [8]. *DirectSound* focused on the capture and playback of digital sound samples, while *DirectMusic* managed message-based musical data. An online article describing the new possibilities of the format can be found in [9].

During its lifetime, such a format was used by many composers and programmers. A non-comprehensive list of video games, which includes major publishers, can be found in a dedicated discussion forum dating back to 2002 [10].

Nevertheless, the lifetime of the *DirectMusic* API was relatively short. In 2006, only 10 years after its official release, it became deprecated. Consequently, not only *DirectMusic*-compatible video-game music was no more produced, but the already existing compositions utilizing such a technology quickly met a fate of decline and abandonment. The goal of this work is to preserve, revive, and pass down this heritage.

3.2 Data Structures

The file encoding is based on the Resource Interchange File Format (RIFF) bitstream format [11], employing data structures known as *chunks* for storing data. A *chunk* is a fragment of information that contains a header carrying some parameters (e.g., the type of chunk, its size, etc.) followed by a variable area containing the data payload. The structure of a RIFF file is very simple: it can be represented as a tree, where each element (*chunk*) has an id, a dimension, possibly a type, and a payload. Unlike a normal tree, where each node has a list of pointers to its subnodes and subnodes reside in a separate area, in RIFF each *chunk* contains the *subchunks* in its payload. The file is therefore presented as a single root (or top-level) chunk that contains all the others, and the header of this *chunk* is also the header of the file. Many multimedia file formats, such as PNG, IFF, MP3, WAV, and AVI, are *chunk*-based.

In *DirectMusic*, the main structures contained in the RIFF file are known as *Forms*, whose most common types are:

- *Segment*, the container for an entire musical excerpt;
- *Style*, carrying the information on patterns that can be dynamically combined at run time;
- *Track*, a list of music events or data to control the performance;
- *Band*, used to define an instrument;
- *Reference List*, a mechanism to connect either different files or *Forms* of a file.

A *Segment* represents an entire piece of music. From a technical point of view, it is a chunk with its id set to RIFF. A *Segment* is usually stored in a separate file, with the extension .sgt (for playback only) or .sgp (for editing with *DirectMusic Producer*). Within the segment, there is a header, which indicates how the song should be played (start and end timestamps, loops, etc.), and the payload is a list of one or more *Tracks*.

The concept of track in *DirectMusic* is more generic than in MIDI: there are many types, and, despite being grouped

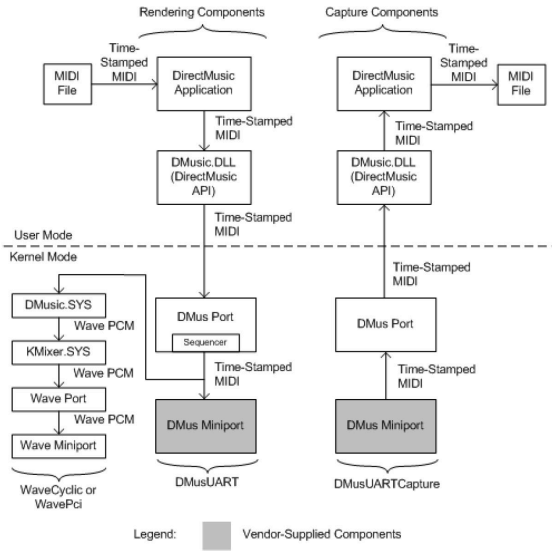

Figure 1. The structure of the *DirectMusic* API. [12].

under a common umbrella, each type has different behavior. Thus, for a parser, it is fundamental to check the *fccType* field in the header. Among the most important *Track* types, it is worth mentioning objects that are simple lists of events (*seqt*), those that give information on tempo changes (*tetr*), those that execute commands that can alter the reproduction (*cmnd*), and those that contain reusable styles and patterns (*sttr*). All *Tracks* are read simultaneously during playback, and events, commands, tempo changes, etc. have an absolute timestamp that defines when they should occur.

One of *DirectMusic*'s strengths is native support for dynamic music, a composition generated on the fly based on the current situation in the game. This result is achieved with the use of *patterns*, namely short musical sections that are automatically selected, combined, and transposed on the chords chosen by the composer. The fundamental parameter that controls such a process is called *groove*, i.e. a number that should give a measure of the intensity of the situation: for example, a high value of groove is typical in a battle or in the fight against the final boss. The groove level can also be varied within the song itself, with a command given by a *cmnd* track: in this case, the aim is simply to build variations of music themes automatically, so as to ease the composer's work.

3.3 Technical Remarks

The structure of the *DirectMusic* API is shown in Figure 1

As it concerns sound synthesis, music can be performed either in hardware, using the Microsoft GS Wavetable SW Synth, or in a custom synthesizer. Thus, audio tracks are rendered by a virtual synthesizer that allows for unlimited channels and rich control messages, expanding the concept of discrete Control Change available in MIDI.

Concerning virtual musical instrument programs, *DirectMusic* adopts DLS [13], a set of standardized file formats

	DMP	*sgt2wav*	*Ficedula*	*libdmusic*
Licence	proprietary	free	unknown	free
Runtime	DirectX	DirectX	DirectX	independent
Operating System	Windows 32 bit	Windows	Windows	Windows/Unix
DM Support	complete (play and modify)	complete (only play)	complete (only play)	subset of DirectMusic 8
Status	abandoned	abandoned	abandoned	under development
Retrieval	waveforms and MIDI subset of sound events	waveforms	waveforms	waveforms

Table 1. Comparison between software solutions that support *DirectMusic*.

for digital musical instrument sound banks developed first by the Interactive Audio Special Interest Group (IASIG), and then by the MIDI Manufacturers Association (MMA). DLS is a versatile format, which in addition to samples, can also define synthesized instruments and supports transformations such as envelopes. The sound font is connected to the *Segment* through a *Form* of type *Reference List*; the single instrument is instead identified through a set of *Band Track* and *Band*. Although a documented standard, the DLS format is no longer widespread today, and software support is also limited, especially on Linux. For this reason, in a context of preservation, it is preferable to convert DLS sound fonts into the universally supported SF2 format.

4. THE DIRECTMUSIC CONVERTER

In this section, we will first review some already existing solutions to convert *DirectMusic* music pieces, and then we will describe our proposal.

4.1 Background

There are a number of solutions currently available to render and export original *DirectMusic* files, but they all have specific drawbacks or limitations. A first distinction concerns the methods that rely on Microsoft's runtime and those that don't. Almost all software that uses *DirectMusic* access it through the Software Development Kit and the runtime provided by Microsoft within DirectX. This means that their functioning is inextricably linked to Microsoft's choice to maintain support and backward compatibility. Furthermore, this aspect precludes the use of *DirectMusic* on operating systems other than Windows. This category embraces, in addition to game engines, plugins that allow you to listen to *DirectMusic* pieces through general-purpose media players (an example is a plugin for Winamp developed by user *Ficedula* from the Final Fantasy community), dedicated reproducers/converters such as *sgt2wav*, and *DirectMusic Producer*.

DirectMusic Producer (DMP) is the original software provided by Microsoft for editing. Last updated in 2002, nowadays even finding a working copy of the software can become a challenge, and it will most likely require a legacy installation of Windows. *DMP* allows exporting a rendering of the performance in a WAV file at the cost of the original "structure" of the document. On the contrary, it is possible to capture MIDI events during a performance.

Another software is *sgt2wav*, a command-line application for Windows. Written in C++, it just requires the DirectX runtime and an audio device. A rendering of the performance is exported with the same limitations as *DMP*. This software will also cease to work when Microsoft will discontinue the *DirectMusic* support in DirectX.

There are also a few independent projects that try to reimplement the format support from scratch, covering operations from parsing to rendering. They are unrelated to support policies, a specific operating system, and version, but, in general, they offer a lower level of compatibility. The only significant example of this category is *libdmusic*, an Open Source (MIT License) library that rewrote from scratch the support for the *DirectMusic* format. For this reason, it is limited to a subset of the functionalities available in version 8. Like the other solutions mentioned here, its goal is to provide a rendering of the output.

A comparison between the mentioned approaches is presented in Table 1.

4.2 dmproj

Our proposal for extracting and preserving information from the *DirectMusic* format relies on software developed in Python 3.x, called *dmproj*. The source code is available on Zenodo under GNU General Public License v3.0. [1]

The software has been conceived with a modular architecture:

- The *riffparse* module recursively parses a RIFF file in order to extract the chunks and their (potential) subchunks.
- The *dmparse* module receives in input a chunk and exposes the underlying structure (e.g., a *Segment*);
- The *dm2midi* module, finally, takes care of generating MIDI events from a Segment. It is important to note that there might be the need to generate multiple MIDI messages for a single *DirectMusic* "event" (e.g., control change messages that correspond to a curve described with only one command).

The output of *dmproj* is a Standard MIDI File (SMF) type 1 [14]. There are multiple advantages to adopting such a format. First, even if standardized decades ago, it is still in use, being supported by many media players and often

[1] https://doi.org/10.5281/zenodo.5952340

```
RIFF 42926 DMSG          /* Root RIFF structure, of type segment (.sgt file) */
.segh 24                 /* Segment header */
.guid 16                 /* Global unique identifier, used by the DirectMusic runtime to load the appropriate segments */
.LIST 116 sgdl           /* Segment data list (undocumented) */
..segd 104
.vers 8                  /* File version */
.LIST 148 UNFO           /* Generic metadata (name, comment, etc) for the whole file */
..UNAM 18
..UCMT 110
.LIST 42562 trkl         /* Track list */
..RIFF 236 DMTK          /* Chord track */
...trkh 32               /* Track header (specifies the track type) */
...LIST 172 cord         /* List of chords */
....crdh 4
....crdb 132
....crdt 8
...ctdc 3
..RIFF 72 DMTK           /* Tempo track */
...trkh 32               /* Track header (specifies the track type) */
...tetr 20               /* List of tempo changes and corresponding timestamp */
..RIFF 1110 DMTK         /* Events track */
...trkh 32               /* Track header (specifies the track type) */
...LIST 30 UNFO          /* Generic metadata (name, comment, etc) for the track */
....UNAM 18
...seqt 876              /* Events container */
....evtl 744             /* Note events */
....curl 116             /* Control events */
...LIST 136 psql         /* Proprietary undocumented structure used internally by DMP. Doesn't affect the actual music */
....psqc 48
....cvau 4
....cvsu 8
....cvsu 8
....cvsu 8
....cvsu 8
..[...]                  /* Other events tracks (usually one per voice/channel) */
..RIFF 28300 DMTK        /* Band track (external encapsulation) */
...trkh 32
...RIFF 28248 DMBT       /* Band track (actual) */
....LIST 28236 lbdl       /* Band list */
.....LIST 776 lbnd        /* Band (external encapsulation) */
......bdih 4
.....RIFF 752 DMBD       /* Band (actual) */
.......guid 16           /* Global unique identifier */
.......LIST 24 UNFO      /* Generic metadata (name, comment, etc) for the band */
........UNAM 12
.......LIST 684 lbil     /* Instrument list */
........LIST 242 lbin    /* Instrument */
.........bins 40         /* Instrument properties (allowed range, channel, volume, transposition, etc) */
.........LIST 142 DMRF   /* Soundfont reference */
..........refh 20
..........guid 16
..........date 8
..........name 20
..........file 18
..........vers 8
.........jzfr 32
........[...]            /* Other instruments */
.....[...]               /* Other bands */
```

Figure 2. Commented dump of a *DirectMusic* music piece (excerpt).

used also for interchange purposes. In addition, with respect to WAV files, SMFs are much lighter, as they contain commands to instruct a synthesizer in producing audio instead of containing audio samples. This approach can also represent a drawback since the final performance is deeply influenced by the quality of audio components (in particular, the MIDI synth) and the availability of high-quality sound samples. Nevertheless, the representation of a performance in terms of commands instead of a fixed, pre-calculated sequence of samples is closer to the original approach of *DirectMusic*. Finally, MIDI performances are not burdened by performing rights.

Conversion of all *DirectMusic*'s features to MIDI requires going beyond the limits of the single standard MIDI file. For example, a music piece could employ more than 16 channels or include waveforms encapsulated directly or through an external reference. For the former problem, a possible solution is to divide the *DirectMusic* song into several MIDI files based on blocks of 16 channels (or 15, if you want to respect the convention that reserves Channel 10 for percussion). To overcome the latter problem, you can export the waveforms in a sound font and insert the suitable control-change and program-change commands in the event flow. Despite these limitations, much of the original information can be kept, thus offering a much more adequate recovery tool than the alternatives.

4.3 Example

In order to demonstrate the efficacy of the proposed solution, we have converted a .sgp file into a .mid file. Both files are publicly available, [2] together with a sound font to be loaded in the MIDI synth for reconstructing the original sounds.

Figure 2 shows the dump of the original music piece, limited to an excerpt for the sake of brevity. Some comments have been introduced to make data structures clearer to the reader.

In this case, the composition is based on a *Segment* only, thus the conversion can be complete.

The difference in size between the original (41.9 KB) and

[2] https://lim.di.unimi.it/media/smc2022_examples/cityfast.zip

the destination file (6.7 KB) shows that the .sgp format was not very efficient at encoding information.

5. CONCLUSIONS AND FUTURE WORKS

In this contribution, we approached the problem of the preservation of video-game music both from a technical and technological point of view and from a philosophical one. Answering the questions of *what and how* we are going to preserve this cultural heritage informed us on the technical decisions that have to be made in the implementation stage.

We looked at the case study of *DirectMusic*, a format that had certain popularity but is now extinct. We presented its characteristics and developed software that allows for a conversion from this format to a currently supported one (MIDI).

The natural development of this project consists in supporting all *DirectMusic* features. In this sense, the main directions are:

1. The extension to the next version of the format, the one used in conjunction with DirectX 8 and 9. Version 9 does not introduce really significant changes, so the two versions can be considered essentially the same. The main changes concern the addition of new fields at the bottom of some data structures. It would therefore be necessary to compare the headers supplied together with the related SDKs to understand the differences and expand the related classes within the decoder;

2. The full support of styles and patterns, which are currently implemented as stubs. A known limitation of the software is the ability to convert only sequence-based music pieces, which can be (and actually are) transformed into MIDI without loss of information. In order to extend its potential, the software should implement the random choice of the pattern to be played, the setting of the groove level, and the transposition through chords. In this way, the piece would be converted in only one of its possible forms since MIDI does not manage the variations. A solution could be to generate all the possibilities;

3. The management of more atypical types of curves used in control messages, such as sinusoidal ones. Even if little used in real games, they are still part of the format and allow interesting creative possibilities.

After improving the conversion tool, the next step will be to organize an archive that allows these heterogeneous materials to be stored, searched, and experienced by users.

While the software presented here is just a first step in a bigger effort, we hope it will elicit a conversation around the topic of the preservation of the digital heritage of video games music.

Acknowledgments

This project has been partially supported by the contribution of the Marshall University Faculty Senate Research Committee (FSRC) Funding.

6. REFERENCES

[1] K. Collins, "An introduction to the participatory and non-linear aspects of video games audio," *Essays on sound and vision*, pp. 263–298, 2007.

[2] S.-S. Chen, "The paradox of digital preservation," *Computer*, vol. 34, no. 3, pp. 24–28, 2001.

[3] H. M. Gladney, "Principles for digital preservation," *Communications of the ACM*, vol. 49, no. 2, pp. 111–116, 2006.

[4] T. Owens, *The theory and craft of digital preservation*. Johns Hopkins University Press, 2018.

[5] A. Baratè, G. Haus, and L. A. Ludovico, "State of the art and perspectives in multi-layer formats for music representation," in *Proceedings of the 2019 International Workshop on Multilayer Music Representation and Processing (MMRP 2019)*. IEEE CPS, 2019, pp. 27–34.

[6] A. Baratè, L. A. Ludovico, D. A. Mauro, and F. Simonetta, "On the adoption of standard encoding formats to ensure interoperability of music digital archives: The IEEE 1599 format," in *DLfM '19: 6th International Conference on Digital Libraries for Musicology*. ACM, 2019, pp. 20–24.

[7] Microsoft, "Microsoft ships DirectX 5.0," https://news.microsoft.com/1997/08/04/microsoft-ships-directx-5-0/, 1997, online; accessed 23 January 2022.

[8] T. Buttram, "DirectX 9 audio exposed: Interactive audio development, chap. beyond games: Bringing DirectMusic into the living room," 2003.

[9] T. Hays, "DirectMusic for the masses," *Gamasutra.com*, 1998.

[10] B. Lynne, S. Bottcher, S. Maloney, S. Morgan, J. Kaae, and D. Yackley, "Games that use DirectMusic," https://www.freelists.org/post/directmusic/Games-that-use-DirectMusic, 2002, online; accessed 23 January 2022.

[11] IBM Corporation and Microsoft Corporation, "Resource Interchange File Format," in *Multimedia Programming Interface and Data Specifications 1.0*, 1991, http://www.tactilemedia.com/info/MCI_Control_Info.html, online; accessed 23 January 2022.

[12] Microsoft, "DirectMusic API Documentation," https://docs.microsoft.com/en-us/windows-hardware/drivers/audio/midi-and-directmusic-components, online; accessed 30 March 2022.

[13] S. J. Welburn and M. D. Plumbley, "Rendering audio using expressive MIDI," in *Audio Engineering Society Convention 127*. Audio Engineering Society, 2009.

[14] MMA, *The Complete MIDI 1.0 Detailed Specification*. Los Angeles, CA: MIDI Manufacturers Association (MMA), 1996.