SUPEROM: A SUPERCOLLIDER CLASS TO GENERATE MUSIC SCORES IN OPENMUSIC

Claudio Panariello
Sound and Music Computing group
KTH Royal Institute of Technology,
Stockholm, Sweden
claudiop@kth.se

Emma Frid
IRCAM STMS Lab,
Paris, France
KTH Royal Institute of Technology
Stockholm, Sweden
emmafrid@kth.se

ABSTRACT

This paper introduces SuperOM, a class built for the software SuperCollider in order to create a bridge to OpenMusic and thus facilitate the creation of musical scores from SuperCollider patches. SuperOM is primarily intended to be used as a tool for SuperCollider users who make use of assisted composition techniques and want the output of such processes to be captured through automatic notation transcription. This paper first presents an overview of existing transcription tools for SuperCollider, followed by a detailed description of SuperOM and its implementation, as well as examples of how it can be used in practice. Finally, a case study in which the transcription tool was used as an assistive composition tool to generate the score of a sonification – which later was turned into a piano piece – is discussed.

1. INTRODUCTION

Automatic generation of notation is a complex topic [1]. The design of computational algorithms to convert acoustic music signals into some form of music notation, so-called Automated Music Transcription (AMT), is a challenging task in signal processing and artificial intelligence [2]. Typically, AMT systems take an audio waveform as input and compute a time-frequency representation, after which a representation of pitches over time is outputted [2]. This process involves, among other subtasks, multi-pitch estimation (MPE), onset and offset detection, beat and rhythm tracking, interpretation of expressive timing and dynamics, as well as score typesetting. A comprehensive overview of signal processing methods for music transcription was presented in [3]. Discussions of challenges for AMT have been published in [4, 2]. While the technological aspects of AMT tools are highly relevant to the work presented in this paper, they differ somewhat from Computer-Aided Composition (CAC) tools (software that allows composers to design computer processes that generate musical structures and data [5]) in the sense that they are primarily designed to aid the transcription process, not necessarily the composition process, which is the purpose of the `SuperOM` presented in this paper. More specifically, the purpose of the current work is to translate code into scores, not to generate scores directly from audio files [1].

Sound synthesis tools used in assisted composition usually do not include automatic music notation features, mainly because these tools were not designed with that particular use case as a primary motivation. However, having the possibility to visualize sounds in standard Western music notation can be useful in many composition contexts. Different attempts have been made to fill this gap, but the efforts have often been characterized by a lack of documentation, making it difficult to present a full review of the tools and methods used. A reoccurring strategy in this context seems to be to use SuperCollider [2] [6] classes that can bridge with LilyPond [7], either directly or through third-party software. SuperCollider is a programming language for audio synthesis and algorithmic composition. LilyPond [3] is a free system to write music.

One of the oldest attempts to bridge sound synthesis software with automatic notation tools is LilyCollider [4], developed by Bernardo Barros [8]. LilyCollider is an interactive software that can build sequences of music notation in an interactive way, extending the SuperCollider programming language (sclang). LilyCollider wraps the LilyPond music notation software, meaning that it can be used to generate a LilyPond score from SuperCollider code. However, the system has some limitations when it comes to rendering time; it requires you to wait until the score is engraved [9]. Today, LilyCollider has been abandoned in favor of SuperFomus [5], which was also developed by Barros. SuperFomus relies both on LilyPond and FOMUS [6] to generate a music score. FOMUS is an open-software application developed by David Psenicka which allows the automation of many musical notation tasks for composers and musicians. It was designed with composers who work with algorithms and computer music software languages in mind and facilitates the process of creating professionally notated scores. An interesting aspect of SuperFomus

[1] Although that would be possible using the SuperOM, for example using spectral analysis features.
[2] https://supercollider.github.io/
[3] https://lilypond.org/
[4] https://github.com/smoge/LilyCollider
[5] https://github.com/smoge/superfomus
[6] https://fomus.sourceforge.net/

is that it allows for musicXML export. In other words, it can be used to generate a file that easily can be edited in any scorewrite software. However, SuperFomus has some limitations, specifically when it comes to more advanced rhythm algorithms, and it may not be the optimal solution for working with metric structures [9]. In other words, it can be somewhat unreliable when dealing with complex music notation structures. In addition, it appears as though is no longer maintained, which may add some troubleshooting time when installing it.

Another system is Fosc [7], which stands for FO-rmalised S-core C-ontrol (FO-r S-uperC-ollider). Fosc is an API that ports much of the Python code base of Abjad [8], a system that supports composers in building complex pieces of music notation in iterative and incremental ways, to SuperCollider. Since Abjad wraps the LilyPond music notation package, Fosc can be used for the generation of musical notation in LilyPond. Despite being powerful, Fosc does not allow for musicXML export, thus limiting the quantity and quality of information that can be preserved in the score.

Finally, a custom-made SuperCollider class called SonaGraphLily was included in the SonaGraph framework, a harmonic spectrum analyzer suitable for assisted and algorithmic composition, developed by Andrea Valle [10]. SonaGraphLily manages mapping from sonographic data (i.e. spectrum over time) to music notation, using LilyPond code. It creates LilyPond source files that are rendered as graphic files. However, besides the fact that this class is optimized to work on a SonaGraph instance, it doesn't support musicXML export.

2. MOTIVATION

Although the above-described software solutions may be useful for certain use cases, they all fall short when it comes to generating musically complex scores while allowing for musicXML export, which is the main aim of the `SuperOM`, described in the forthcoming sections. In other words, the goal of `SuperOM` is to enable generation of scores from SuperCollider in the fastest way possible with as few dependencies as possible, while at the same time preserving extremely high precision in the notation, and allowing for musicXML export.

The `SuperOM` is a SuperCollider class that produces music scores in the form of OpenMusic (OM) files, i.e. in the form of `.omi` files. OpenMusic [11] is an open-source environment [9] dedicated to music composition, developed at IRCAM in the end of the 1990s (see e.g. [12, 13]). It is a visual programming language based on Lisp allowing the user to design processes for the generation and manipulation of music material, but it can also be used for other applications [13]. Similarly to other graphical environments such as Pure Data (Pd) [10] and Max/MSP [11], the workflow to code programs in OpenMusic is based on patching together different modules. However, as opposed to such tools, the output of the OM processes is also visualized using conventional music notation. A screenshot of the OpenMusic interface is displayed in Figure 1. OpenMusic has been used by a wide community of composers and computer musicians throughout the years. Notable composers include, among others Kaija Saariaho, Marco Stroppa, Brian Ferneyhough, Philippe Manoury, Fabien Lévy, and Mauro Lanza. [12]

There are several reasons why OpenMusic was adopted in this project. Firstly, OpenMusic's powerful capabilities, especially in terms of handling very complex music structures, as well as its exporting features, were important features. OpenMusic allows for export into many different file formats, e.g. MIDI, bach [14], and musicXML. As such, it enables the generation of files that can be opened and edited in most common scorewriter software (such as MuseScore, Sibelius, Finale, or Dorico, just to name a few). Another reason is the simplicity of the overall installation: `SuperOM`'s only dependency is, in fact, OpenMusic, which installation is quite straightforward. Moreover, no prior knowledge of OpenMusic is actually required, since the only OpenMusic features required by a SuperCollider user are import/export file. Finally, yet another reason is the aim to bridge two open-source software solutions as well as their communities. In fact, `.omi` files generated by the `SuperOM` in SuperCollider are completely legit and working OpenMusic files. This means that they can be manipulated directly by OpenMusic users. This aspect can encourage collaborative frameworks in which SuperCollider and Open Music users can work together, exchanging their own material.

However, as for all software solutions, the adoption of OpenMusic might have some drawbacks. For example, the playback functionality in OpenMusic could be considered an obstacle, since it requires a third-party software synth player to work, which is out of the scope of this paper. Nevertheless, it seems that future versions of OpenMusic will have an embedded synth available. [13] On the other hand, if a SuperCollider user decides to use OpenMusic merely as a way to export a musicXML file for a notation software, the playback part can indeed be skipped.

3. CLASS DESCRIPTION

The main method of `SuperOM` is `.writeOMfile`, which takes the following arguments: `fileName`, `midicents`, `magnitudes`, `rhythmTree`, `metronome`, `quantization`, `threshold`, `dynamics` (see Listing 1).

Once an `.omi` file has been produced, it can be imported in an OpenMusic patch and opened from there. The file can then be edited directly in OpenMusic and exported as an XML file (for example using the POLY object).

[7] `https://github.com/n-armstrong/fosc`

[8] `https://abjad.github.io/`

[9] It can be downloaded for free from `https://openmusic-project.github.io/openmusic/`

[10] `https://puredata.info/`

[11] `https://cycling74.com/`

[12] `https://en.wikipedia.org/wiki/OpenMusic`

[13] Please see the OpenMusic IRCAM Forum `https://discussion.forum.ircam.fr/t/open-music-midi-player/39930`, accessed 14 December 2022.

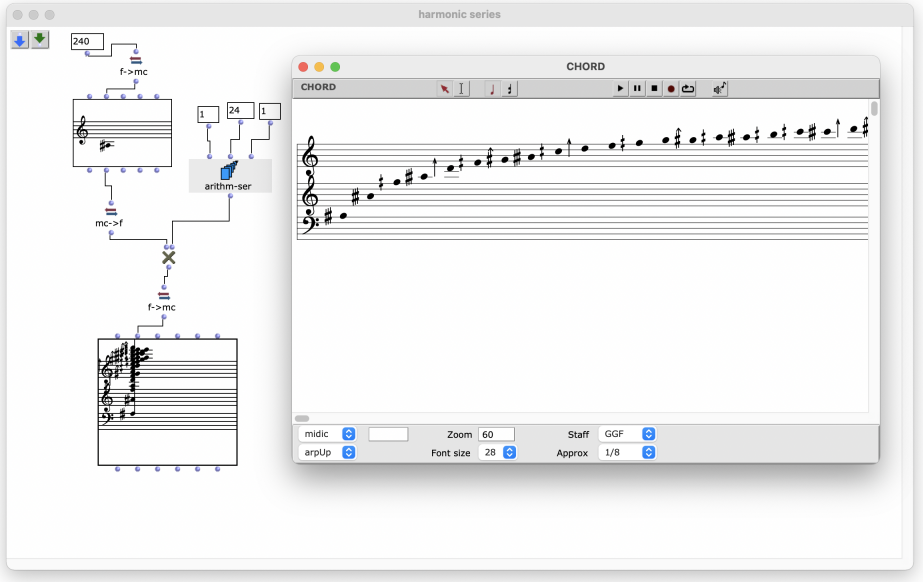

Figure 1. A basic example of an OpenMusic patch. In this example, several modules are connected to generate a harmonic series on the fundamental frequency 240, up to the 24th harmonic. The window CHORD is displaying the result using conventional music notation.

```
o = SuperOM.new;
o.writeOMfile(fileName:, midicents:,
    magnitudes:, rhythmTree:, metronome
    :, quantization:, threshold:,
    dynamics:);
```

Listing 1. SuperOM

A more detailed description of the arguments taken by `.writeOMfile` follows below.

3.1 fileName

The argument `fileName` is the name for the output file, including the `.omi` extension (e.g. `"test.omi"`). The output will be produced in the same folder where the SuperCollider file is located.

3.2 midicents

The argument `midicents` is an array of notes, expressed in midicents. This argument is always passed as an array of staves, thus leaving to `SuperOM` the task to interpret how many staves should be written in the OM output file. For example, in the case of $[[l_0, l_1, l_2, ...], [m_0, m_1, m_2, ...], [n_0, n_1, n_2, ...], ...]$ the rows are interpreted as subsequent separated staves; the first staff will contain notes $[l_0, l_1, l_2, ...]$, the seconds staff will contain notes $[m_0, m_1, m_2, ...]$, the third staff will contain notes $[n_0, n_1, n_2, ...]$, etcetera. Chords can be specified with additional brackets, e.g. $[[n_0, n_1, [n_2, n_3], n_4, ...]]$. In this example, there is one staff and the notes n_2 and n_3 are interpreted as a bichord.

3.3 magnitudes

The argument `magnitudes` is an array (or an array of arrays) of the notes' magnitudes. `magnitudes` can be expressed either in decibels (dBs) or as MIDI velocities (i.e. values from 0 to 127), the `SuperOM` will take care of interpreting the given array in the correct way with the method `.dbvelSmart`. `magnitudes` must match the size of `midicents`. If no magnitudes are given, all the notes will automatically be set to velocity $= 100$.

3.4 rhythmTree

`rhythmTree` is an array of rhythms, provided as floats (or fractions), i.e. 0.5 or $1/4$ for a quarter note, 0.25 or $1/8$ for an eight note, etc. A positive value represents the duration of a note, while a negative value represents a pause. For example, $-1/16$ represents a rest of a sixteenth note. A rest, however, can also be specified using the SuperCollider's operand `Rest()`. If this argument is `nil`, the `magnitudes` argument will be used as a source to create a rhythm tree (see Section 4.2).

3.5 metronome

This argument specifies the metronome of the score. If only one value is given, the metronome will be the same for the entire score. As an alternative, an array of metronomes can be specified, matching the rows size of the `midicents` array, thus assigning a different metronome to each staff. The default metronome value is 60 bpm.

3.6 quantization

Amount of quantization of the notes, expressed in MIDICENTS. The default value is 50 (i.e. a quarter-tone).

3.7 threshold

This argument, expressed in dB, sets a threshold for the magnitudes, meaning that values below this level will be

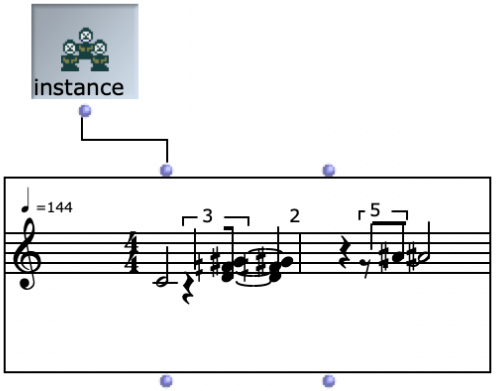

Figure 2. Instance of an OpenMusic POLY class. The figure shows the content of the file produced by the code shown in Listing 2.

considered as silence, i.e. pauses. The default value is −36 dB.

3.8 dynamics

The dynamics flag can be set to `true` or `false`. If `true`, the output file will display the notes' magnitudes as music dynamics (i.e. from "*ppp*" to "*fff*") in the score. The MIDI velocities are converted into music dynamics using the method `.veldyn`. The default value is `false`.

4. EXAMPLES

In the following section, a number of examples are provided to demonstrate potential use cases for `SuperOM`. All of the example .omi files, as well as the corresponding XML output, are available for download from here: `https://tinyurl.com/9j2fba5p`.

Listing 2 shows the most basic example of `SuperOM`: the array `pitches` contains a list of notes and a chord, everything expressed in midicents; the array `rhythm` contains a list of durations and pauses. Line 5 of Listing 2 produces an `.omi` file with the name `"example0.omi"` that will contain the music material specified in the two arrays. The result is a file that, once imported in OM, will look like the one displayed in Figure 2.

```
1 var pitches = [6000, [6200, 6550,
    6800], 7000];
2 var rhythm = [1/2, -1/6, 2/6, -2/5,
    3/5];
3
4 o = SuperOM.new;
5 o.writeOMfile("example0.omi", [pitches
    ], rhythmTree: rhythm, metronome:
    144);
```

Listing 2. Basic usage of SuperOM.

Listing 3 shows another simple usage of the class, this time producing a chromatic scale with eighth-tones, starting from C4, as 32th notes. Please notice that in order to create a score with eighth-tones, we have to specify the correct quantization in `writeOMfile`, using `quantization: 25`.

```
1 var pitches = (6000, 6025..7200);
2 var rhythm = {1/32}.dup(pitches.size);
3
4 o = SuperOM.new;
5 o.writeOMfile("example1.omi", [pitches
    ], rhythmTree: rhythm, metronome:
    144, quantization: 25);
```

Listing 3. A chromatic scale with eighth-tones starting from C4.

Listing 4 shows a simple variation of the previous example, adding random magnitudes and printing them in the score, with the flag for the argument `dynamics` set to `true`. The result imported in OM will look like the one displayed in Figure 3.

```
1 var pitches = (6000, 6025..7200);
2 var mags = {rrand(-18, -3)}.dup(
    pitches.size);
3 var rhythm = {1/32}.dup(pitches.size);
4
5 o = SuperOM.new;
6 o.writeOMfile("example2.omi", [pitches
    ], magnitudes: mags, rhythmTree:
    rhythm, metronome:144, quantization
    : 25, dynamics:true);
```

Listing 4. A chromatic scale with eighth-tones starting from C4, with dynamics.

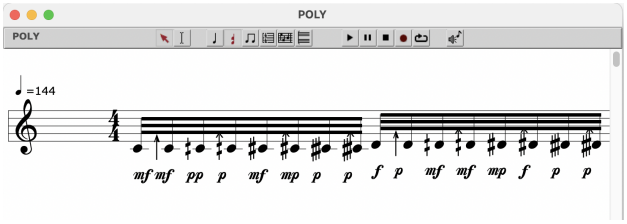

Figure 3. Instance of an OpenMusic POLY class showing the content of the file produced by the code shown in Listing 4.

Using standard features provided in SC, a more complex score can be generated with a few lines of code. Listing 5 shows the code to produce a score with the following properties: five staves with different chromatic scales; rhythms and pauses chosen by a given set, with each staff having a different metronome.It is worth noticing that in this example the argument pitches is passed without extra brackets (see line 7) since it has already been created as an array of staves (see line 2). The result imported in OM will look like the one displayed in Figure 4.

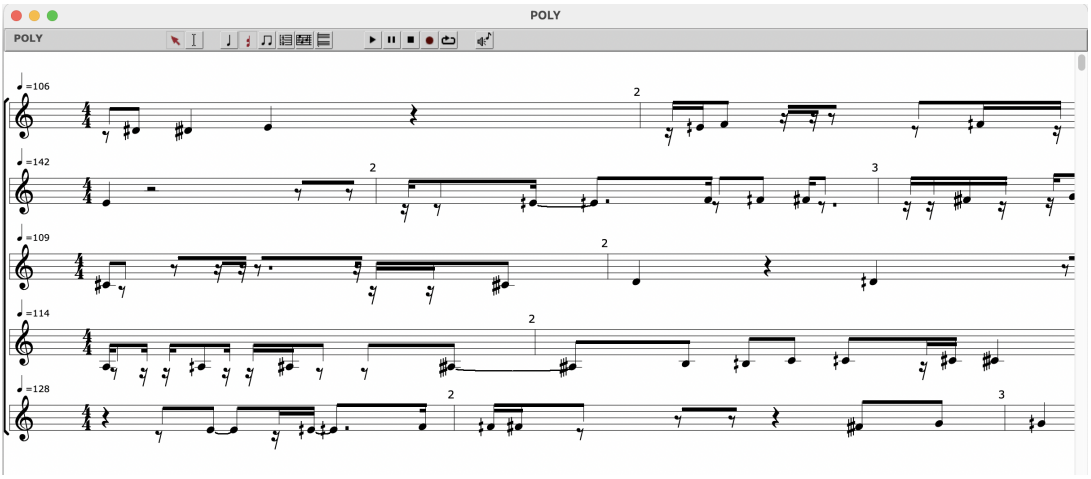

Figure 4. Instance of an OpenMusic POLY class showing the content of the file produced by the code shown in Listing 5.

```
var staves = 5;
var pitches = {(6000, 6050..7200)+(
    rrand(-5, 5)*100)}.dup(staves);
var rhythm = {{[1/4, 1/8, 1/16].choose
    *[-1, 1].choose}.dup(pitches.shape
    [1]*2)}.dup(staves);
var metronomes = {rrand(102, 144)}.dup
    (staves);

o = SuperOM.new;
o.writeOMfile("example3.omi", pitches,
     rhythmTree: rhythm, metronome:
    metronomes);
```

Listing 5. Snippet of code to produce a score with five staves with different chromatic scales, rhythms with pauses chosen by a given set, and five different metronomes.

Listing 6 shows code that produces a score with eight staves, each of them containing random frequencies taken from a harmonic series, random magnitudes, and random rhythms, quantized to 32th notes. The frequencies are conveniently translated into midicents thanks to the `.cpsmidicents` method.

```
var notes = 200, staves = 8;
var pitches = {{Array.fill(24, {|i|(i
    +1*rrand(50, 51))}).choose}.dup(
    notes)}.dup(staves).cpsmidicents;
var mags = {{rrand(-18, -3)}.dup(notes
    )}.dup(staves);
var rhythm = {{rrand(0.1, 1).softRound
    (1/32, 0, 1)}.dup(notes)}.dup(
    staves);

o = SuperOM.new;
o.writeOMfile("example4.omi", pitches,
     mags, rhythm, 84, 25, -36, true);
```

Listing 6. Snippet of code to produce a more complex score with eight staves.

4.1 Writing a score from patterns

Patterns are typical SuperCollider data structures that allow for the management of events in time, specifying the rules for the production of such events [1]. As demonstrated in Listing 7, a musical piece expressed through a pattern can then be translated into a score using the `SuperOM`. In order to do that, events must be conveniently stored into separate arrays, for example using the method `.collect`. These can then be used as arguments for generating the output score.

```
var length = 50, pitches, rhythm;
p = Pbind(
        \midinote, Pxrand([60, 62, 64,
            66, 68, 70], inf),
        \dur, Prand([1/16, 1/8, Rest
            (1/16), Rest(1/8)], inf));

e = p.asStream;
pitches = length.collect({e.next(()).
    midinote})*100;
rhythm = length.collect({e.next(()).
    dur});

o = SuperOM.new;
o.writeOMfile("example6.omi", [pitches
    ], rhythmTree: rhythm, metronome
    :144);
```

Listing 7. Writing a score from a pattern.

4.2 Writing a score from spectral data

Another interesting application of `SuperOM` is the ability to write a score from spectral data, as seen in Listing 8. This feature can be useful in situations when you want to create a score from a series of frequencies and magnitudes from a spectral analysis process. The strategy used here is based on grouping subsequent notes that have the same frequency and magnitude. Grouping in this context means that rhythmic values are summed together (for example,

two $1/8$ notes are replaced by one $1/4$ note). Notes that have different magnitudes will not be grouped. Magnitudes below the given threshold value are interpreted as silence, meaning that such notes are transformed into pauses.

```
var freqs = {rrand(400, 500)}.dup(50).
    cpsmidicents;
var mags = {[-18, -12, -9, -6].choose
    }.dup(freqs.size);
freqs.postln;
mags.postln;
o = SuperOM.new;
o.writeOMfile("example5.omi", [freqs],
     magnitudes: mags, rhythmTree: nil,
     metronome:144, quantization: 100,
    threshold: -12, dynamics:true);
```

Listing 8. Writing a score from spectral data

5. CASE STUDY – GENERATING A PIANO PIECE USING SUPEROM

The SuperOM has been intensively used by the first author in his artistic practice. The tool has been very useful especially since it allows for the creation of transcriptions of material derived from spectral analysis. Below is a short account of a case study in which SuperOM was used in a composition process involving sonification of video material.

SuperOM was successfully used in a joint study carried out together with the pianist and researcher Johan Fröst. This project made use of sonification of a video that was created from multiple video recordings of Fröst playing Debussy's *"Reflects dans l'eau"*. More specifically, the aim of the project was to sonify moving images using a Disklavier piano, a self-playing piano designed by Yamaha [14] . Sonification is defined as the use of nonspeech audio to convey information [15] [15] .

The video to be sonified was created starting from multiple video recordings of Fröst's performance. The videos were edited together, highlighting musical events and the musical narrative of Debussy's piece. The final merged video was then sonified, and the sonification was used as starting material in the composition process. The aim of the composition process was to create a new piano piece to be played in synch with the original video in a live performance at a concert series in the Spring of 2023. The incpit of piano piece is shown in Figure 5.

The overall sonification workflow involved three stages and two different softwares: 1) the video was loaded into Max/MSP, in which video processing took place using, among others, the cv.jit package [16] ; 2) the data was sent to SuperCollider via OSC [17] , where the mappings for the actual sonification was implemented; 3) MIDI messages were generated in SuperCollider and sent to the Disklavier, which played the newly generated piano piece. To establish a strong connection between the video and the generated sound, the sonification used harmonic content derived from the same Debussy piece that drove the creation of the video.

Once the sonification was realized, SuperOM was used to generate an actual score of it. In order to do that, the mappings relative to pitches, note durations and velocities were stored into separate arrays, and used to initialize an instance of an SuperOM. The output score was an accurate representation of the sonification: as a matter of fact, the complex metric structures produced with the sonification were completely captured in the score. Once exported in musicXML, the file could be opened without errors in a commercial music notation software, where the composition process continued by selecting and merging musically interesting materials, and by reducing their complexity in order to make them playable. However, this workflow had some limitations. Firstly, despite the score containing all the note velocity information, thus making its MIDI playback sound correctly, it did not have any music dynamics printed on the staves. This was solved by improving the code and adding the dynamics flag presented in Section 3. A second obstacle was that the score produced by the SuperOM contained a large number of staves (namely 25), resulting from how the sonification data was stored. This aspect made the score quite impracticable to read and difficult to work on, and it required a staves merging operation, done by hand, in order to achieve a typical piano-looking score. In the future, such a problem could be solved by carefully designing SuperOM methods to collapse many staves into one.

It should be noted that the final product of the process outlined above was not a literal sonification mapping the video input directly to output (i.e. the final piece) in an objective way; the author interacted with the generated material, merging and adapting different generated parts into a final piece, thus disrupting the direct connection between input and output. In other words, sonification was used as a subtask within the composition process, assisting the author's composition process by providing ideas for the final piano piece. This process is somewhat similar to what is referred to as mixed-initiative interaction in the field of Human Computer Interaction (HCI), in which each agent (human or computer) contributes what it is best suited at the most appropriate time [16]. Using this mixed strategy outlined above, the piece closes the circle of re-mediation from a piano piece by Debussy, to: 1) a video made with the intent to visualize the musical material of Debussy's piano music; 2) a sonification based on this video, using material from the original piano music; 3) the use tonal material created in step 2), which in turn served as material for a new composition for the piano; and finally 4) a performance played in real-time by a pianist.

[14] https://www.disklavier.com/

[15] This definition has later been expanded by Thomas Hermann to: *"(...) data-dependent generation of sound, if the transformation is systematic, objective and reproducible (...)*, see https://sonification.de/son/definition/.

[16] https://jmpelletier.com/cvjit/

[17] https://ccrma.stanford.edu/groups/osc/index.html

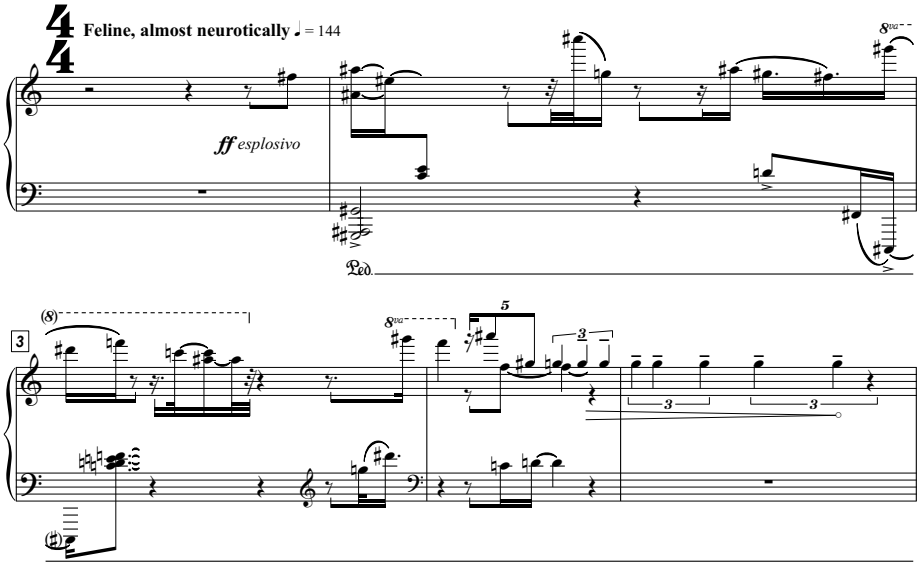

Figure 5. Excerpt from the beginning of the piano piece composed starting from a sonification of video images.

6. LIMITATIONS

A limitation of the current version of the `SuperOM` concerns combining midicents staves of different lengths. As a matter of fact, midicents arrays should all have the same length, in order to create a final rectangular matrix. One workaround to solve this issue is to fill the shorter arrays with zeros (a sort of zero padding), thus matching the size of the longest array, see Listing 9. In this way, the additional 0 pitches will be ignored, as long as the rhythm tree doesn't contain rhythm information.

```
var pitches1 = [7200, 7400, 7500, 7600];
var rhythm1 = [1/6, -2/6, 1/4, 1/4, 1/4];

var pitches2 = [6000, 6200, 6550, 6800, 7000, 6800, 5300, 5625, 6378, 6740];
var rhythm2 = [1/2, -1/6, 0, 0, 2/6, -2/5, 3/5, 1/6, -1/6, 0, 0, 1/6, 1/4];

var pitches3 = [5500, [5600, 5950], 5700, 6050];
var rhythm3 = [-1/8, 1/8, 1/8, 1/8];

pitches2.do({pitches1=pitches1++0});
pitches2.do({pitches3=pitches3++0});

o = SuperOM.new;
o.writeOMfile("example9.omi", [pitches1, pitches2, pitches3], rhythmTree: [rhythm1, rhythm2, rhythm3], metronome: 144);
```

Listing 9. Generating a score from midicents arrays with different lengths.

Interestingly, rhythm trees that contain zeroes make subsequent notes collapse, thus creating chords. This effect is demonstrated in the example in Listing 10, in which two ways of writing the same musical score are compared. As before, also here we need to zero pad the shortest arrays.

```
var pitches1 = [6000, [6200, 6550, 6800], 7000, 6800, [5300, 5625, 6378], 6740];
var rhythm1 = [1/2, -1/6, 2/6, -2/5, 3/5, 1/6, -1/6, 1/6, 1/4];

var pitches2 = [6000, 6200, 6550, 6800, 7000, 6800, 5300, 5625, 6378, 6740];
var rhythm2 = [1/2, -1/6, 0, 0, 2/6, -2/5, 3/5, 1/6, -1/6, 0, 0, 1/6, 1/4];

pitches2.size.do({pitches1=pitches1++0});

o = SuperOM.new;
o.writeOMfile("example10.omi", [pitches1, pitches2], rhythmTree: [rhythm1, rhythm2], metronome: 144);
```

Listing 10. Comparison of two different ways of producing the same score in output.

7. CONCLUSIONS AND FUTURE WORK

This paper presents `SuperOM`, a class for bridging SuperCollider with OpenMusic thus enabling generation of complex music scores with a high level of precision. Files generated with `SuperOM` can be imported and edited in OpenMusic, which allows for collaborative frameworks between the two software.

The SuperCollider code of the implementation of `SuperOM` is still in progress and is continuously improved. The code is readily available on from here: `https://github.com/claudiopanariello/SuperOM`. The material provided includes the `SuperOM` files and a tutorial file. At the point of writing, the first author is using `SuperOM` in several projects, for example in the generation of scores from algorithmic compositions realized in SuperCollider, or in the creation of music transcriptions of material derived from spectral analysis of audio recorded material (especially audio feedback recordings). This ongoing work will continue to inform the design of `SuperOM`, allowing it to be iteratively improved over time. The use case presented in Section 5 serves as a formative evaluation of the `SuperOM`, carried out by the author from a first-person perspective. In the future, there is a need for empirical evaluation with actual users, to identify weaknesses and areas of possible improvement.

It is worth mentioning that the music scores discussed in this paper refer to traditional Western score. There are many situations in which other score notations might be more appropriate. Therefore, a possible future direction might be to design other classes that could allow for non-standard notations, similarly to what Ghisi and Agostini did in extending bach by introducing the dada library [17].

Acknowledgments

The first author would like to express his gratitude to Mauro Lanza, who kindly helped out with the OpenMusic code.

8. REFERENCES

[1] A. Valle, *Introduction to SuperCollider*. Logos Verlag Berlin GmbH, 2016.

[2] E. Benetos, S. Dixon, Z. Duan, and S. Ewert, "Automatic music transcription: An overview," *IEEE Signal Processing Magazine*, vol. 36, no. 1, pp. 20–30, 2018.

[3] A. Klapuri and M. Davy, *Signal processing methods for music transcription*. Springer Science & Business Media, 2007.

[4] E. Benetos, S. Dixon, D. Giannoulis, H. Kirchhoff, and A. Klapuri, "Automatic music transcription: Challenges and future directions," *Journal of Intelligent Information Systems*, vol. 41, no. 3, pp. 407–434, 2013.

[5] D. Bouche, J. Nika, A. Chechile, and J. Bresson, "Computer-aided composition of musical processes," *Journal of New Music Research*, vol. 46, no. 1, pp. 3–14, 2017.

[6] S. Wilson, D. Cottle, and N. Collins, *The SuperCollider Book*. The MIT Press, 2011.

[7] H.-W. Nienhuys and J. Nieuwenhuizen, "LilyPond, a system for automated music engraving," in *Proceedings of the XIV Colloquium on Musical Informatics (XIV CIM 2003)*, vol. 1. Citeseer, 2003, pp. 167–171.

[8] B. Barros, "LilyCollider and rhythmic structures," *Revista Vórtex*, vol. 2, no. 2, 2014.

[9] Bernardo Barros, "Music Notation with SuperCollider." [Online]. Available: http://bernardobarros.com/files/lilycollider-sc2013/slides.pdf

[10] A. Valle, "SonoGraph. a cartoonified spectral model for music composition," in *Proceedings of the 16th Sound & Music Computing Conference*. SMC, 2019, pp. 462–469.

[11] G. Assayag, C. Rueda, M. Laurson, C. Agon, and O. Delerue, "Computer-assisted composition at IRCAM: From PatchWork to OpenMusic," *Computer Music Journal*, vol. 23, no. 3, pp. 59–72, 1999.

[12] C. Agon, "Openmusic: Un langage visuel pour la composition musicale assistée par ordinateur," Ph.D. dissertation, Paris 6, 1998.

[13] J. Bresson, C. Agon, and G. Assayag, "OpenMusic: Visual programming environment for music composition, analysis and research," in *Proceedings of the 19th ACM international conference on Multimedia*, 2011, pp. 743–746.

[14] A. Agostini and D. Ghisi, "A Max library for musical notation and computer-aided composition," *Computer Music Journal*, vol. 39, no. 2, pp. 11–27, 2015.

[15] G. Kramer, B. Walker, T. Bonebright, P. Cook, J. H. Flowers, N. Miner, and J. Neuhoff, "Sonification report: Status of the field and research agenda," University of Nebraska - Lincoln, Tech. Rep., 2010.

[16] J. E. Allen, C. I. Guinn, and E. Horvtz, "Mixed-initiative interaction," *IEEE Intelligent Systems and their Applications*, vol. 14, no. 5, pp. 14–23, 1999.

[17] D. Ghisi and A. Agostini, "Extending bach: A family of libraries for real-time computer-assisted composition in Max," *Journal of New Music Research*, vol. 46, no. 1, pp. 34–53, 2017.