Trajectory Score Library ：

A tool for algorithmic spatialisation with Antescofo

nadir B.
Spectacle Vivant & Art Sonore
info@nadirbabouri.fr

ABSTRACT

This paper presents Trajectory Score Library, an Antescofo library of scripts, converting mathematical parametric functions into curves in order to control the Spat5 sources trajectories. These scripts are written in the Antescofo language, a timed synchronous language which allows the use of relative time to synchronize the trajectories with musical events. This language can also provide a convenient way of creating and transmitting automation parameters to the Spat5 through the Open Sound Control protocol implemented in Antescofo. Trajectory Score Library is an additional tool amongst existing spatialization solutions. In the context of this presentation these solutions are considered as remote-control software. Trajectory Score Library thus aims to use Max as a unified real-time environment to unite different computer music processes.

Following the paper Trajectory Score Library, that was presented at the JIM[1] in 2020 at Strasbourg, the author would like to introduce recent development that occurred and how he implemented them in the Antescofo Language, as proposed in the perspectives section of the previous article.

1. COMPILATION

An important feature implemented in the antescofo language is the ability to compile user defined functions when the score is loaded in the Antescofo object. This will "…produce an equivalent. But more efficient version of a given function… for that a C++ code is.produced, compiled and linked on the fly with the running Antescofo instance."[2]

It is a very suitable operation in the Trajectory Score Library scripts because they are mainly composed of parametric mathematical functions and thus the compilation will afford smoother mouvements of the audio sources' trajectories.

In the case of this 3D parametric function called *couronne sinusoidale* , where n > 0,

```
@fun_def @couronneX($a,$b,$n,$t,$offsetX){
$ra*@cos($t)+ $offsetX}

@fun_def @couronneY($a,$b,$n,$t,$offsetY){
$a*@sin($t) + $offsetY}

@fun_def @couronneZ($a,$b,$n,$t,$offsetZ){
$b*@cos($n*$t) + $offsetZ}

@compilation(MAP{

        @couronneX -> [["double", "double",
        "double", "double"], "double"]]
        @couronneY -> [["double", "double",
        "double", "double"], "double"]]
        @couronneZ -> [["double", "double",
        "double", "double"], "double"]]}
```

The type of function is represented by a tab with 2 elements[3] :

- The first element is a tab that lists the type of the. arguments (an empty tab if there is no arguments).

- The second element is a type of the returned value.

[1] Trajectory Score Library (Consulted the 22/06/2023)

[2] https://antescofo-doc.ircam.fr/Reference/compilation/#compilation (Consulted the 22/06/2023)

[3] https://antescofo-doc.ircam.fr/Reference/compilation/#compilation (Consulted the 22/06/2023)

The declared *MAP*[4] is a dictionary which associates a value to a key and enables to compile a bunch of functions at once.

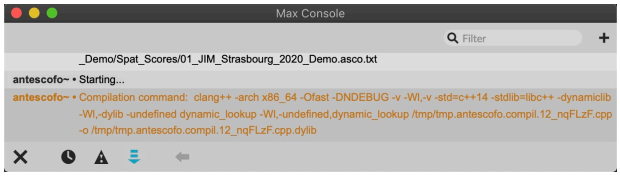

Figure 1 shows a Max console printing the compilation message

2. DEFAULT PARAMETERS

It is now possible to write a process, within antescofo, with default parameters which makes the code more readable. These parameters are declared in the process as follow :

```
@proc_def rectilinear($source=1, $iniT=-1, $targeT=1, $a=0, $b=1, $alpha=1,
$beta=0,$count, $speed){...}
```

And now a simple call to ::rectilinear() will trigger the trajectory with these default parameters.

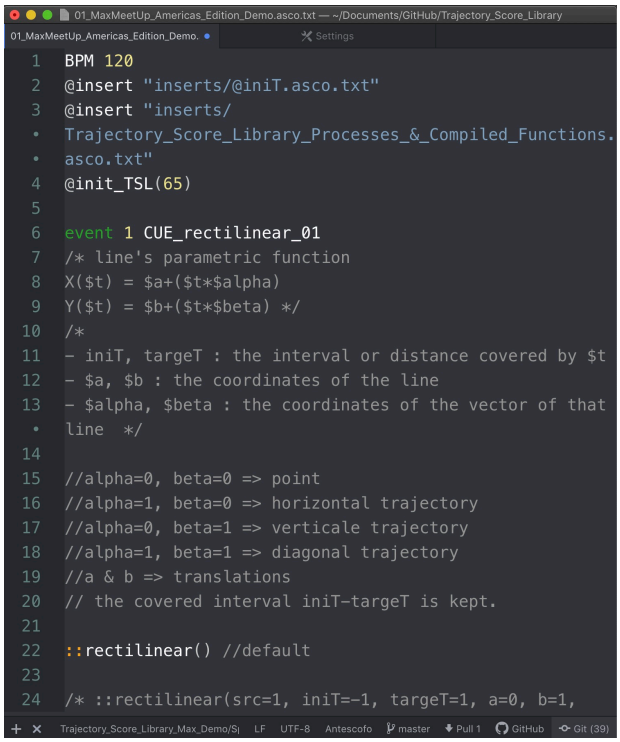

Figure 2 shows a Trajectory Score Library script in the Atom text editor.

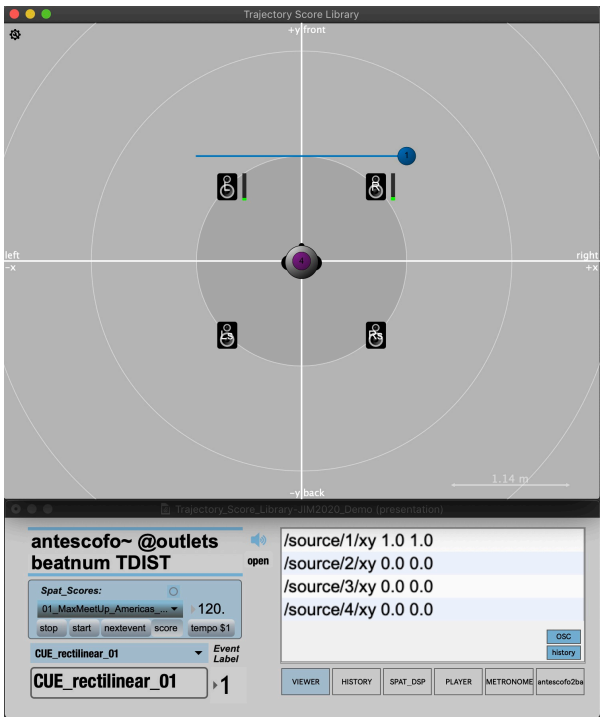

Figure 3 shows the default rectilinear scene in the Spat5 viewer.

Besides these parameters can now be declared in different order affording that they are explicitly written this time without the $ sign :

```
::rectilinear(source=1, speed=1, count=1)
```

In this case, the parameter becomes optional in the process call, as the default value is used in place of the missing argument.

3. CREATING SPATIAL SCENES

As presented in the title, the notion of algorithmic spatialisation is made explicit by the creation of different scenes using predefined functions, macros, processes, and actors available in the Antescofo language[5].

3.1. CLOUD_OF_POINTS SCENE

The initial process used for this demonstration is a simple 2D trajectory *called* ***::rectilinear()*** which will be the basis of a scene called **Cloud_of_points**. It will trigger, when called, a random spatial position for each chosen sound source at once.

[4] https://antescofo-doc.ircam.fr/Reference/data-map/ (Consulted the 22/06/2023)

[5] https://antescofo-doc.ircam.fr/Reference/1-intro/ (Consulted the 22/06/2023)

First, in order to achieve fixed positions in space, the vector coordinates of the line has to be null:

```
@proc_def rectilinear($source=1, $iniT=-1, $targeT=1, $a=0, $b=0, $alpha=0,
$beta=0,$count, $speed){...}
```

In order to produce random positions, a user-defined function generator is first implemented thanks to *@random a* native function of the Antescofo language :

```
@fun_def @rand_range($min, $max){

        @rand($max-$min)+$min }
```

Then implemented in the rectilinear process to control the coordinates of the line :

```
::rectilinear(source=1, iniT=-1, targeT=1, count=1, speed=1, alpha=0, beta=0,
a=@rand_range(-1, 1), b=@rand_range(-1, 1))
```

Each time this process will be triggered, a random position will be applied to audio source 1 and around the listener in a maximum distance of 1 meter.

In order to produce multiple random positions of a unique source, in a certain amount of time or duration, the previous process can be imbeded in a loop construction. A loop function is afforded in the program language [6]

```
loop random_points 1 {
        ::rectilinear(...)
} during[4#]
```

This process will trigger random postions during 4 beats, related to the BPM declared at the beginning of the score. To have a better control over the duration and choose to which audio source the process is applied, this loop can also be embedded in another process to declare additional variable parameters :

```
@proc_def ::random_points_relative($src,$sync,$dur)
{
 loop random_points_rel $sync
 {
        ::rectilinear($src, a=@rand_range(-1, 1),
        b=@rand_range(-1, 1), alpha=0, beta=0,
        speed=$sync)
 } during[$dur#]

::random_points_relative(3, 6, 4)
```

Launching this process will apply 4 random positions to source 3, with an interval of 6 beat between each position. The **$sync** variable will link the periodicity of the loop to the overall scene in order to avoid unnecessary jumps of the source. And a *during* clause is used to stop the loop[7]. Declaring the same process and setting the *during* clause of the loop in milliseconds - **during[$dur ms]** - is useful for the spatialisation of an audio file, synchronised to its duration.

Accordingly, a cloud of points, with multiple sources, can be achieved by either creating a group body[8], that allows to gather common actions and tie them to a declared tempo[9],

```
group cloud_of_points @tempo=60
{
 ::random_points_relative(1, 1, 8)
 ::random_points_relative(2, 1, 8)
 ::random_points_relative(3, 1, 8)
 ::random_points_relative(4, 1, 8)
}
```

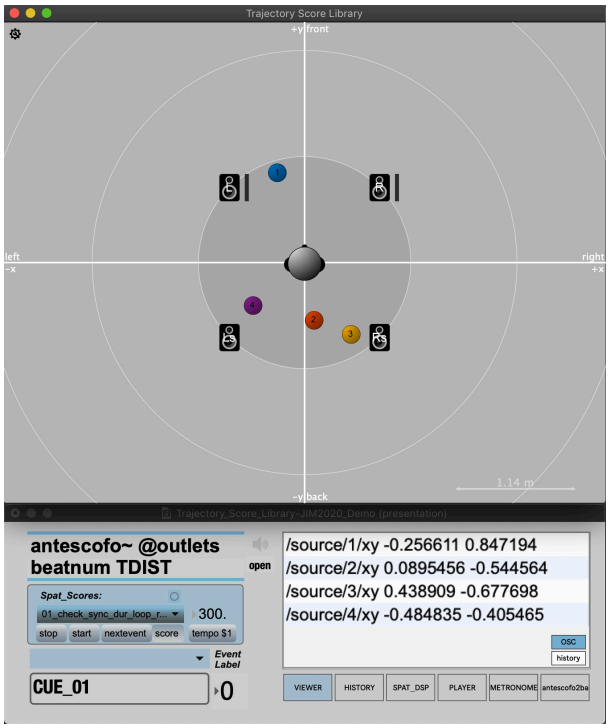

Figure 4 shows a capture of one state of the cloud of points scene.

[6] https://antescofo-doc.ircam.fr/Reference/compound_loop/ (Consulted the 22/06/2023)

[7] https://antescofo-doc.ircam.fr/Reference/compound_loop/#stopping-a-loop (Consulted the 22/06/2023)

[8] https://antescofo-doc.ircam.fr/Reference/compound_group/ (Consulted the 22/06/2023)

[9] It is possible to modulate this tempo with a user-defined function.

It can also can be achieved by declaring a macro definition[10] that will allow to parameterize fragments of code by a variable name like in the following scene.

3.2. THE SQUARE SCENE

This time a macro definition is used to program a square scene with controlled speed of the audio source's movement, at each side of the square, during the trajectory. The default line process is used. The coordinates of the line and those of the vector are hard coded. The only parameter that is tweaked through the macro is the duration or speed of the movement of the source. This same duration is set for the delay between each trajectory or action of the process.

```
@macro_def @square($dur1,$dur2,$dur3,$dur4{
::rectilinear(a=-1, b=1, alpha=1, beta=0, $dur1)
$dur1
::rectilinear(a=1, b=1, alpha=0, beta=-1, $dur2)
$dur2
::rectilinear(a=1, b=-1, alpha=-1, beta=0,$dur3)
$dur3
::rectilinear(a=-1, b=-1, alpha=0, beta=1,$dur4) }
```

Once this *macro* is declared, a simple call @square with the four duration parameters : @square_01(4,1,1,1/4) will trigger the prepared scene. Actually, the macro definition can be stored in an external antescofo file and loaded at start time, just like the file containing the trajectory processes. This method will provide a more readable score with clear and distinct lines of code.

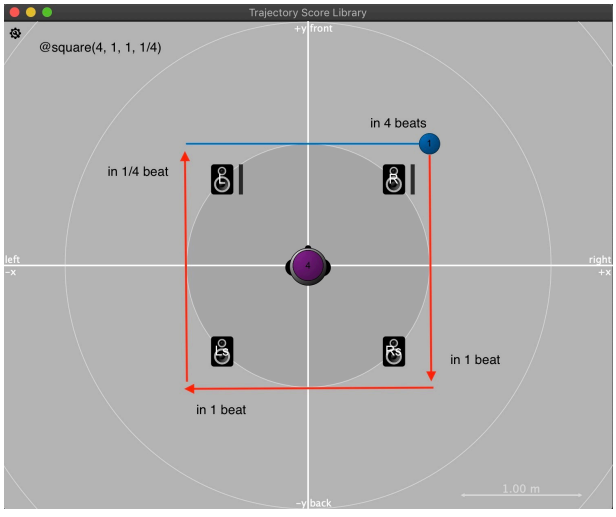

Figure 5 shows a square scene.

4. SPATIALISATION PER NOTE BASIS I

In the following demonstration, the previous square scene, with controlled duration of the movement of the sound source, is used for the spatialisation of a musical instrument per note basis and the synchronisation of these notes with the movement in a vey simple manner.

Let us say we have four successive quarter notes and would like the sound of each note to cross exactly, in the same duration as the note, each side of the square surrounding the listener. The score will be then written this way :

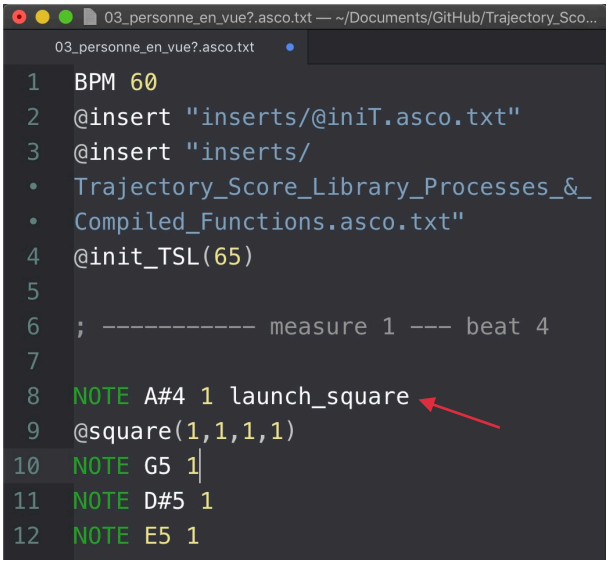

Fig. 6 shows a symbolic representation converted into an antescofo score

Where the square scene is declared right after the first note[11], with the trajectory duration parameter set to the same durations of the notes.

In this Antescofo or electronic score, a tempo of 60 is declared at the beginning through the *BPM* instance. Then two files are called, at load through the *@insert* function. The first file *@iniT.asco.txt* contains Max and Antescofo initiation commands and some user-defined functions. The second one loads the parametric mathematical functions, that are compiled at load, and the different trajectory processes that the user may use in his electronic composition. These two files are loaded when the message *start* is sent to the antescofo object.

[10] https://antescofo-doc.ircam.fr/Reference/11-macros/#macro-definition-and-usage (Consulted the 22/06/2023)

[11] If the listening machine were on, it will wait for this note to be played in order to launch the action.

Then a first event is declared with the *NOTE* specification, which is a container of one pitch specified with the midi A#4, its duration of 1 beat - related to the. *BPM* above - that is a quarter note in this case. A label may be used in order to follow the score on the Max interface, which is very handy during rehearsals.

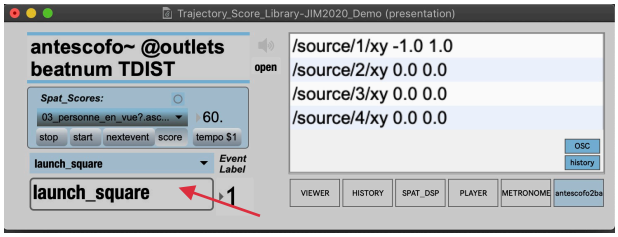

Figure 7 shows the Trajectory Score Library Max interface with the printed label of the first event in the loaded score.

The square scene will be triggered as soon as the A#4 event is either played or detected by the listening machine[12]. If ever we apply another rhythmic pattern, let's say a triplet followed by a dotted half note, and would like to synchronise the same spatialisation scene with this musical phrase, we simply apply those rhythmic divisions to the duration parameters of the macro :

NOTE A#4 **1/3** launch_square
@square(1/3, 1/3, 1/3, 3)
NOTE G5 **1/3**
NOTE D#5 **1/3**
NOTE E5 **3**

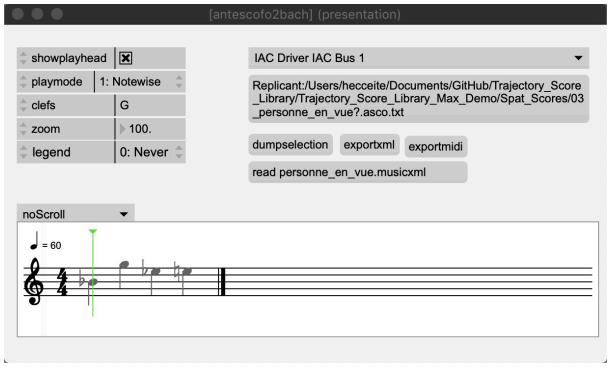

Figure 8 shows an implementation of a bachroll receiving the score from the antescofo

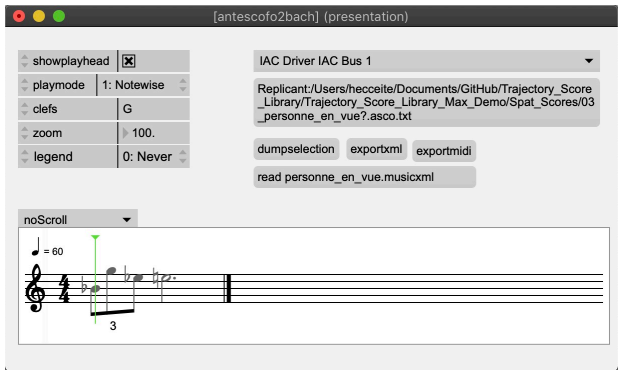

Figure 9 shows an implementation of the bachroll. Object updated with the score from the antescofo object[13]

Besides, changing the tempo of the score through the BPM declaration do not request to rewrite all the durations through the score. Antescofo does the translation. It is the case for complex subdivisions or nested rhythmic patterns.

4.4. THE DOT NOTATION

The reference of an instance of a process can be used to assign a variable local to a process from the outside. And then address a specific parameter of a process with the dot notation[14], that will transform its state after the first run[15] :

```
$linear_var := ::rectilinear(src=1, count=2, speed=4)
4
$linear_var.$speed := 16
```

This example will trigger a line mouvement of the audio source number 1, a trajectory that will last four beat. Once the target position achieved the source will go through the same trajectory but this time four times slower.

[12] https://antescofo-doc.ircam.fr/UserGuide/electronic/#antescofo-as-a-sequencer (Consulted the 22/06/2023)

[13] https://antescofo-doc.ircam.fr/Library/Functions/bach_score/ (Consulted the 22/06/2023)

[14] https://antescofo-doc.ircam.fr/Reference/exp_variable/#the-dot-notation (Consulted the 22/06/2023)

[15] In order to be effective the process must run a second time to take into account the transformation i.e. the process must be in the loop mode by increasing the $count parameter of the process.

5. CONCLUSION & PERSPECTIVES

The author has presented some implementations of the Antescofo language both in the scripts that makes use of mathematical parametric functions to control audio sources' spatialisation, and through some algorithms that allows the construction of spatial scenes, stored and called in the electronic score. These developments have occurred, between the first proposition of the paper to the Journée d'Informatique Musicale in march 2020 at Strasbourg, and its presentation and publication at the JIM.

6. REFERENCES

1. Bascou C. « Adaptive Spatialization and Scripting Capabilities in the Spatial Trajectory Editor Holo-Edit. » Proceedings of the 7th Sound and Music Computing Conference, Barcelona, 2010.
2. Baskind A. Quelques notions sur la spatialisation. Ircam - Centre Pompidou, Formation Spatialisateur, Paris, 2008.
3. Bresson J. et Schumacher M. « Representation and Interchange of Sound Spatialization Data for Compositional Applications. » Proceedings International Computer Music Conference, Huddersfield, UK, 2011.
4. Bresson J. Spatial Structures Programming for Music. Spatial Computing Workshops (SCW), Valencia, Spain, 2012.
5. Bresson J., Bouche D., Carpentier T., Schwarz D., and Garcia J. « Next-generation Computer-aided Composition Environment: A New Implementation of OpenMusic. » Proc. of the International Computer Music Conference (ICMC), Shanghai, China, Oct 2017.
6. Carpentier T. « ToscA: An OSC Communication Plugin for Object-Oriented Spatialization Authoring. » Proc. of the 41st International Computer Music Conference, Denton, TX, USA, Sept. 2015. pp 368-371
7. Carpentier T. « A new implementation of Spat in Max. » In Proc. of the 15th Sound and Music Computing Conference (SMC), Limassol, Cyprus, July 2018. pp. 184–191
8. Cont A. « ANTESCOFO: Anticipatory Synchronization and Control of Interactive Parameters in Computer Music », International Music Conference (ICMC), Aug. 2008, Belfast, Ireland. pp.33-40. Hal-00694803.
9. Echeveste J., Giavito J.-L. et Cont, A. A Dynamic Timed-Language for Computer-Human Musical Interaction. [Research Report] RR-8422, INRIA. 2013.
10. Garcia J., Carpentier T., and Bresson J. Interactive-compositional authoring of sound spatialization. Journal of New Music Research – Special Issue on Interactive Composition, 46(1):74 – 86, 2017.
11. Gottfried R. and Bresson J. « Symbolist: An Open Authoring Environment for End-user Symbolic Notation. » International Conference on Technologies for Music Notation and Representation (TENOR'18), Montreal, Canada, 2018.
12. Jacquemin G., Coduys T. et Ranc M. « Iannix 0.8. » Actes des Journées d'Informatique Musicale, Mons, Belgique, 2012.
13. Puckette M. « Combining Event and Signal Processing in the MAX Graphical Programming Environment. » Computer Music Journal, vol. 15, no. 3, 1991.