ADAPTIVE SPATIALIZATION AND SCRIPTING CAPABILITIES IN THE SPATIAL TRAJECTORY EDITOR HOLO-EDIT

Charles Bascou
GMEM
Centre National de Création Musicale
15 rue de Cassis
Marseille, France
`charles.bascou@gmem.org`

ABSTRACT

This paper presents recent works on controlling and editing sound spatialization on multiple speakers based on sound descriptors. It has been implemented as an extension of Holo-Edit, an OpenSoundControl compliant multitrack spatial trajectory editor developed at GMEM. An SDIF interface has been implemented allowing importing and visualizing sound descriptors generated by third party softwares. A set of scripting tools is proposed to process and map these time-tagged data to sound trajectory generation.

1. INTRODUCTION

Sound spatialization has become an important field in the past decades. From Karlheinz Stockhausen early experiments of moving acoustic sound sources around a set of microphones to contemporary cinematographic multichannel mastering, there has been a lot of growing interests in experimenting with sound diffusion across multiple loudspeakers. We can distinguish two main approaches in the work with sound in space. One has been the GRM's Acousmonium [1], initiated by Francois Bayle in 1974 which tends to spatialize stereo tracks manually from a mixing desk onto a loudspeaker orchestra. In its main principle, using an eclectic set of speakers allows the sound to be spatialized by itself, even with no intervention from the electroacoustic music performer. The acoustic characteristics of the different loudspeakers and their position in the concert hall makes them unique sound sources with their specific color and *timbre*, some enhancing high frequency, some medium ones, etc. Movements of sound are then created obviously by electroacoustic music interpretation but also by the *movement* and the *energy* of the sound itself. The other main approach has been the virtual acoustic model where sound spatialization is performed by mathematical and acoustical laws of sound in space. One is for example the distance cue which is simulated by attenuating sound volume (roll-off), filtering high frequencies (air absorption) and increasing the reverberation ratio of the spatialized sound. Here, movements of sound is done by controlling the virtual source position with calculated trajectories or via external input devices such as joystick or graphics tablet.

The focus of this paper is between these two models, in other words to get virtual acoustic movements of sound closer to the sound properties and behaviors. Our software environment, Holo-Edit, is a graphical editor for spatialization trajectories and is particularly adapted for controlling virtual acoustic DSP softwares. Our main goal here is to enhance trajectory editing and spatial cues by taking into account the inner structure and dynamic profiles of the sound to be spatialized. This is using the principle of adaptive audio effects [1] applied to sound spatialization. We will first detail the main features of the used environment Holo-Edit, then detail our motivations and finally present the proposed adaptive spatialization framework.

2. HOLO-EDIT FEATURES

Holo-Edit is initially part of the HoloPhon project initiated in 1996 by Laurent Pottier at GMEM [2] [2]. This project was focused on sound spatialization editing and control. Growing computing power allowed to develop custom DSP spatialization softwares written in MaxMSP, under the generic name Holo-Spat. For now, we will detail Holo-Edit features, as it is the main environment for our experiments.

2.1 Workflow

Holo-Edit is a standalone application written in Java/Jogl. The main underlying paradigm is the control of external DSP softwares via the OSC protocol. In [3], we showed the benefits of a stratified approach in sound spatialization environments. In this scheme, Holo-Edit has its place as an authoring tool for composing with space. All DSP processes are handled in other layers, e.g. in external softwares like MaxMSP, PureData, SuperCollider, etc. Holo-Edit only deals with movement of sound in space, using the timeline paradigm found in traditional DAWs to record, edit, and play back control data. Although a straightforward OSC protocol has been defined in order especially to

[1] Groupe de Recherches Musicales - Paris

[2] Groupe de Musique Experimentale de Marseille

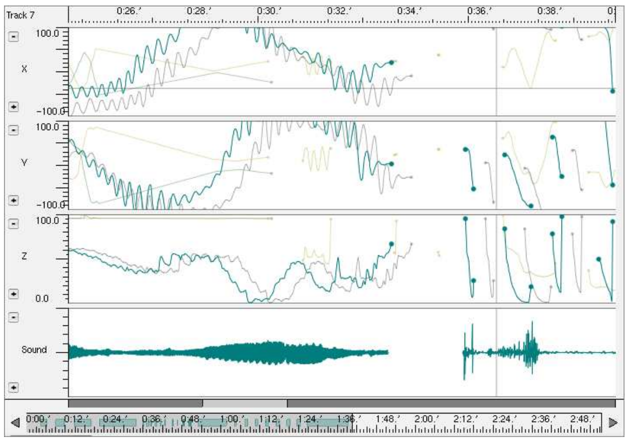

Figure 1. *Time Editor* view with cartesian xyz components and sound waveform.

keep Holo-Edit and the external spatialization software in sync in terms of time region selection and track muting and visualizing, an effort is made to standardize the way Holo-Edit deals with common control messages like source position. As it is the most promising attempt in that purpose, we are currently following the SpatDIF [4] initiative, with first experimental interfaces in the Jamoma Modular environment [5].

2.2 Multitrack Data representation and editing

Holo-Edit manipulates trajectory objects which is a set of time-tagged 3D points. These trajectories have an onset and offset time. Various graphical editing function can be achieved on these object like stretch, extend, trim, join. Maximum time resolution for the trajectories and points has been set to one millisecond allowing precise sound event/position mapping.

Additionally, audio waveforms can be imported from traditional sound files, and included in the composition. Sound cues are then triggered in parallel with the trajectories. It also helps in synchronizing sound events and corresponding position in space while editing.

Various representations of spatialization data are proposed. The *Room Editor* is a top-view editor of the virtual scene where you can move points and trajectories. The *Time Editor* in Figure 1 focus on time/data representation in a similar way as DAW softwares do for automation curves. The user can view and edit individually cartesian and polar coordinates components as curves. In this view, you can also visualize pre-imported sound waveforms time-aligned with corresponding 3d positions.

The *3d Room* shown in Figure 2 offers a 3d representation of the trajectories in the virtual scene although no editing can be achieved in it.

Holo-Edit is closely inspired by traditional Multitrack DAW. The *Score* view uses the timeline paradigm to represent the whole spatialized composition. Sound blocks and trajectory blocks can be moved and copied from one track to another. Traditional solo and mute functions are also implemented.

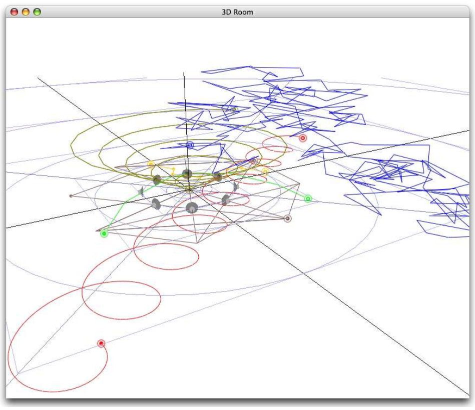

Figure 2. *3D Room* visualization.

2.3 Data Recording via OSC

Trajectories can be generated in various ways. One of them is the ability to record track position in real time from an external program or device via the OSC protocol. Several tracks can be recorded at the same time. For each track, the *segment* message allows to start a new trajectory where all future recorded points will be stored. It is particularly meaningful when using graphics tablet as input device, starting a new trajectory each time the stylus touches the tablet. In this purpose, user can import soundfiles and then, while playing them, record in real time their corresponding trajectories. All the proposed transformation functions can be applied then to smooth, scale or translate the recorded movements.

2.4 Generation/transformation plugins

In the Holo-Edit environment, the ability to process spatially and temporally the spatialization data is an important feature. Graphical editing could be a good help but sometime lacks accuracy for precise and repetitive tasks. A plugin interface has thus been defined for generative and transformative functions. They share a common scheme in their application. Functions are applied into the global time selection. They can output and/or apply the result on one, all or only visible tracks. It allows to make batch process on several tracks and trajectories. There are three process categories : *Generative Functions*, *Spatial Transformations* and *Temporal Transformations*. *Generative Functions* includes circular, lissajou, brownian and random algorithm. *Spatial Transformations* deals with basic geometrical transformation like rotation, translation, proportion. It also includes some more specific processes like exaggeration which scale the local movement of a trajectory leaving the main form unchanged. In the *Temporal Transformations*, the user can perform time stretch, acceleration or time reverse.

3. MOTIVATIONS

One often uses algorithmic functions to generate movements of a specified sound. In this framework, circular, brownian and random movements are the most commonly used and offer, in their combinations, a wide range of different spatial figures. Though their efficiency, it is not so easy to tune their parameters to fit the spatialized sound behaviors, for example, finding the correct circular speed with an iterative sound (supposing quasi-synchronous iterations). The same problem comes when using random or brownian movements on chaotic granular sound materials where we would wish internal sound events to be placed individually. Quite often it results in ”spatial contradictions” that is when the spatialization movement contradict the sound internal behavior and *energy*. Note that this *energy* is quite a subjective and cultural notion. Instrumental sound is good illustration of this phenomenon. When listening to instrumental sound without any visual stimuli, the felt movement and energy are generally strongly associated with those engaged in the instrumental interpretation of a musician. String instrumental sound for example inspire the back and forth movement of the bow. In this framework, the main idea is to find in sound dynamic characteristics these inner profiles, such profiles helping then in setting the virtual movement in sync with its properties.

In a traditional sound synthesis workflow, it is not common to generate spatialization movements when synthesizing or mixing sound. In the purpose of generating/enhancing movements closely related to the sound inner structure, it is obvious that when working with sound synthesis, we could deduce meaningful time profiles from the different modulations and interactions defined in the synthesis process. Such time profiles could greatly help in creating spatial movements. Unfortunately, they are not so easy to route and store in an efficient way. If the internal control signals of these synthesizer are accessible, MIDI sequencers could be a solution but with a known lack of data and time accuracy.

Moreover, spatialization is still a process which is quite difficult to setup in a home studio environment. This work often takes place in dedicated spatialization studio. This is contributing in making sound generation/mixing and spatialization two processes which hardly come together in the same place and time.

The main idea is then to be able to analyze sound to be spatialized, extracting characteristic profiles in its structure. These profiles can then be mapped to various parameters of the trajectory generation/transformation.

4. ADAPTIVE FRAMEWORK

4.1 SDIF Data import and Visualization

SDIF (Sound Description Interchange Format) [6] is a standard generic, open, and multi platform format for sound description data storage. An SDIF file contains one or more sequences of entities called *frames*. Each Frames are time-tagged and typed among a wide range of defined sound descriptors. For example, let’s cite Fundamental Frequency, Loudness, Noisiness as traditional synchronous data flux associated with sounds. These descriptions can also be asynchronous like transient markers or chord separation markers. In this case data rate is not necessarily constant.

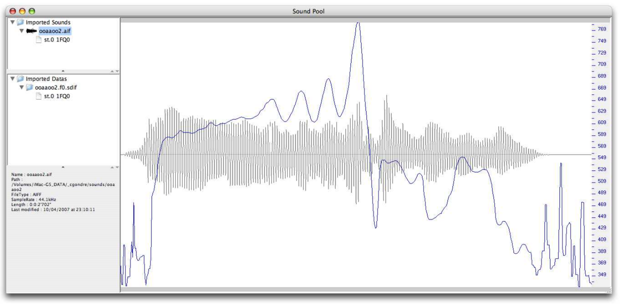

Figure 3. Waveform and its fundamental frequency shown in the *SoundPool* window.

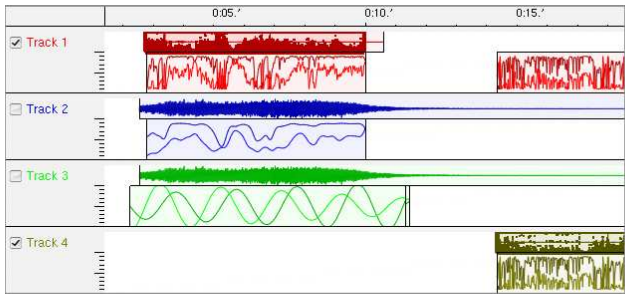

Figure 4. *Score* window with waveform and data visualization on top of the trajectories.

We chose SDIF as it is widely adopted in sound analysis softwares like IRCAM AudioSculpt and AsAnnotation or Michael Klingbeil’s SPEAR [7]. Current SDIF implementation includes a library in C/C++. Since Holo-Edit has been written in java, we had to find a way for interfacing with this standard SDIF library. Thanks to the SWIG interface (Simplified Wrapper and Interface Generator) [8], we could develop a native java wrapper for SDIF. It is composed of a java classes and a platform specific jnilib.

SDIF import in Holo-Edit is done via the *SoundPool* window. As shown in Figure 3, one can overlay sdif data and the corresponding previously imported waveform. These data objects can then be dropped in the *Score* window on a specific track. At this step, these objects are not yet interacting with the spatialization score. They act more as visual cues and markers for enhanced editing especially in the *TimeEditor* window where precise resynchronization of trajectories and points can be achieved.

An important feature is that data objects can be linked with waveform objects so that they are always in sync in the *Score* and the *TimeEditor* window. It is particularly useful when using these data to generate or modify trajectories as we will see in the next section.

4.2 Direct Mapping Plugins

One key feature of the SDIF support is the ability to use these time profiles to generate or transform trajectories. A first step has been to propose a simple plugin interface that converts these time-tagged values into 3d point parameter.

This generative function works in the same scheme as the others functions do. It uses the global time selection to define the region where the new trajectory will be written.

It generates trajectory in the selected tracks. Data access is also done according to the time selection. The plugins scans the score for each overlapping data object instances in the score and then construct a menu on the interface for selection of the data we want to use. A special naming protocol has been defined referencing a data object instance characterized by the SDIF filename, the SDIF stream id, its begin time in the score and its parent track. For example, an fundamental frequency data object instance on track 2 starting at 52 seconds would be named :

```
3-francesca_04.f0.sdif - st.0 1FQ0 - begin time
    =0:0:52:000 - Track:2
```

Note that, as these data objects are in a kind of global scope, the plugin can access them from all tracks. That can avoid duplicating the same data object on multiple tracks when the user wants to generate different mapping of the same data on multiple tracks.

In the cartesian coordinates version of the plugin, the user can assign a different data instance to each XYZ components. As there can be multiple streams of time tagged value in a data object, user has to choose an available one depending on the analysis software which generated the SDIF file. For Example, with fundamental frequency data object (FQ0), AudioSculpt gives us four streams which are *Frequency* in Hz, *Confidence* coefficient, *Score* coefficient and linear *RealAmplitude*. All these values have different unit so a scaling process has to be made. This is done automatically with linear mapping from minimum and maximum of the data values to -1. and 1. for cartesian components. In fact, scaling and translating can be made afterwards with all the graphical and algorithmic functions already available so there is no need to propose scaling feature in the plugin interface.

This direct data mapping can be interesting for simple experiments. It allows to easily transform data points into trajectory points. These profiles can then be shaped in space and time with all the available editing functions like scaling, translation, rotation, etc. But as we will see in the next section, it becomes quite limiting when using more complex data (like multidimensional ones) or experimenting original transformative plugins.

4.3 Script interface

The direct mapping plugins presented in the last section can be really helpful for simple data. But quickly comes the need to have more complex mapping of these profiles. Obviously, in the scheme of adaptive spatialization, this process is central; this is where the user defines the relation between sound characteristics and spatial cues. As it is a very vast field, it is important to let the user the ability to experiment by himself original relations. We thus had to find a workflow as open as possible with a rich expressivity that can well define complex relations and this is what scripting can do. Notably, scripts may be particularly useful in different cases:

1. Performing repetitive tasks efficiently.
2. Applying algorithms with precision on some data.
3. Getting some information about available data.
4. Mathematical functions availability.
5. Creating libraries for trajectory transformations and generations.
6. Algorithms fast experimentations.

Thanks to the *Groovy* project [9] which aims to propose an agile and dynamic language for the Java Virtual Machine, we could build a script interface integrated in the internal Holo-Edit environment. Groovy is built on top of Java and is found to be an easy to learn and powerful object oriented scripting language. Its benefits are multiple : on the fly execution of code from a text buffer or from interactive console, dynamic typing, Java context integration, and so on. The last one is particularly interesting as it easily allows to access to any java classes and functions the main java program uses. In our case, it has greatly simplified the interface with the Holo-Edit Java objects such as 3d points, trajectories and tracks. Moreover an abstract java class has been developed to facilitate scores data access and scripting with them.

The *Script* window presents a text area where the script can be edited. An additional text area is shown called ”Values from score” where useful local variables are proposed. They are *begin*, *end* and *duration* of the global time selection when the script function was called. It also present the name of the available SDIF data instances overlapping the time selection. This name can be copy and paste in the script and is used as a reference for data instances. Getting data instance handle is made for example with :

```
mySDIFdata = getSDIFdata("quat-cel1.f0.sdif -
     st.0 1FQ0 - begin time=0:0:38'213 -
    Track:0")
```

Some helper functions has been included in the interface especially for sampling data at a specific time making transformative script easier when point timestamps of the transformed trajectory have to remain unchanged.

Here is a simple example of transformative script where fundamental frequency is mapped to the z components of the trajectory points :

```
import holoedit.data.*;

f0data = getSDIFdata(gp,
  "3-francesca_04.f0.sdif - st.0
  1FQ0 - begin time=0:0:52'290 - Track:2")
int dateBegin = 5171;
int dateEnd = 74636;
double dur = 69465;

float min = minFieldValue(f0data,0);
float max = maxFieldValue(f0data,0);
float mean = meanFieldValue(f0data,0);
float range = max - min;

HoloPoint point;
int date;

for (int i = 0; i < gp.copyTrack.size(); i++)
{
  point = gp.copyTrack.elementAt(i);
  date = (point.date - dateBegin)*10;
  if( hasDataAtTime(f0data,date))
    point.z = 50 + ( getDataAtTimeField(f0data,
        date,0) -mean ) * 200 / range;
}
```

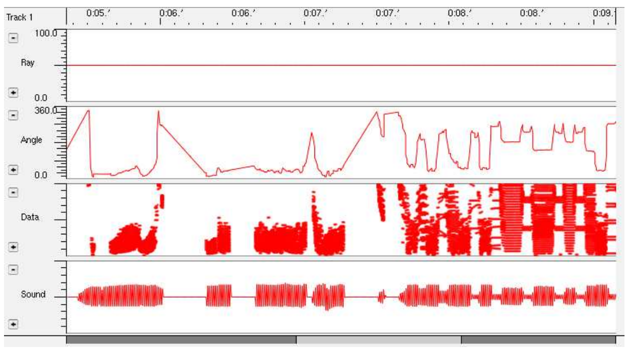

Figure 5. *Time Editor* view of the trajectory generated by brightness mapping.

Another benefit of using script is that the user can generate or transform a trajectory from multiple data instances. For example, he could use voiced/unvoiced segmentation and fundamental frequency to define rules of spatial behaviors; let's say for example random movement for unvoiced part and frequency dependent position for voiced part.

4.4 User case: sound brightness mapping

We will present in this section an example of application of this adaptive framework. We chose to use sound brightness as it has found to be very efficient to discriminate sound events in a complex mix. Its profile contains a lot of relevant informations as it detect timbre abrupt changes. It could be compared to fundamental frequency but with more robust results in a great variety of sound materials (e.g. non harmonic sounds or complex mixture). An interesting feature of sound brightness mapping is that its profile contains good representation of transients and steady parts of the sound.

Moreover, this example is good illustration of data filtering and processing. Indeed, no known end-user software can produce brightness, or spectral centroid its DSP equivalent, SDIF files. We thus had to compute it directly from spectral data. In that purpose, we used a spectral peak analysis to preprocess and simplify the raw spectrogram. The peak analysis outputs for each frame a set of peaks defined by its frequency, amplitude and phase. So we included in the mapping script the computation of an estimate of the spectral centroid, that is the frequency barycenter of the peak set. It gives us a one dimensional data array easily mappable to a trajectory parameter.

The experiment has been to map this value to the azimuth of the source, the distance remaining constant. The data rate, and consequently the point rate, has been set to 5 ms to be able to capture fast timbre changes. The result is kind of spatial magnification of the sound. Each occurring similar event gets its own position in space giving a very coherent spatial image of the sound. The Figure 5 shows the resulting trajectory in the Time Editor with ray, azimuth, data and waveform curves. This technique works great for complex mixture or sound material but harmonic instrumental sounds get poorer results as their *timbre* evolve much slowly. In this case, it should be probably better to elaborate algorithms based on fundamental frequency.

5. CONCLUSIONS AND FUTURE WORK

Adaptive spatialization offers a great field of new possibilities in trajectory generation and transformation. We proposed, to exploit it, an experiment interface in end-user spatialization authoring tool. It has the benefit to allow original editing functions while keeping fast and easy graphical fine tuning of spatial behaviors. We planned some enhancements of the script interface to get to a more global scope on the spatialization score. The generation of trajectories on multiple track for example could help when generating parallel hardly correlated trajectories. Some particular generation and transformation algorithms, especially the one based on sound brightness, will also be implemented has standard Holo-Edit plugins in order to make them available to user impervious to script editing. The Holo-Edit software and sources are available for download on the GMEM website `http://www.gmem.org`.

6. ACKNOWLEDGMENTS

The author would like to specially thank Leopold Frey and Charles Gondre for their great contributions to the development of this project and composer Rodrigo Cicchelli Velloso for his interest in the musical application of the prototyped framework.

7. REFERENCES

[1] X. Amatriain, J. Bonada, A. Loscos, and J. Arcos, "Content-based transformations," in *Journal of New Music Research, 32(1)*, pp. 95–114, 2003.

[2] L. Pottier, "Dynamical spatialization of sound. holophon : a graphic and algorithmic editor for sigma1," in *Dafx98 Proceedings*, 1998.

[3] N. Peters, T. Lossius, J. Schacher, P. Baltazar, C. Bascou, and T. Place, "A stratified approach for sound spatialization," in *Proceedings of The 6th Sound and Music Computing Conference*, 2009.

[4] N. Peters, "Proposing spatdif - the spatial sound description interchange format," in *Proceedings of the 2008 International Computer Music Conference, Belfast*, 2008.

[5] T. Place and T. Lossius, "Jamoma: A modular standard for structuring patches in max," in *Proceeding of the International Computer Music Conference 2006*, 2006.

[6] D.Schwarz and M. Wright, "Extensions and applications of the sdif sound description interchange format," in *Proceedings of the International Computer Music Conference, Berlin*, 2000.

[7] M. Klingbeil, "Software for spectral analysis, editing, and synthesis," in *Proceedings of the International Computer Music Conference, Barcelona*, 2005.

[8] *SWIG*. `http://www.swig.org/`.

[9] *Groovy*. `http://groovy.codehaus.org/`.