Multi-Platform Development of Audiovisual and Kinetic Installations

Iannis Zannos
Ionian University, Dept. of Audiovisual Arts
Plateia Tsirigoti 7
Kerkyra, 49100 Greece
+30 6977280656
zannos@gmail.com

Jean-Pierre Hébert
UCSB, Kavli Institute for Theoretical Physics
hebert@kitp.ucsb.edu

ABSTRACT

In this paper, we describe the development of multi-platform tools for Audiovisual and Kinetic installations. These involve the connection of three development environments: Python, SuperCollider and Processing, in order to drive kinetic art installations and to combine these with digital synthesis of sound and image in real time. By connecting these three platforms via the OSC protocol, we enable the control in real time of analog physical media (a device that draws figures on sand), sound synthesis and image synthesis. We worked on the development of algorithms for drawing figures and synthesizing images and sound on all three platforms and experimented with various mechanisms for coordinating synthesis and rendering in different media. Several problems were addressed: How to coordinate the timing between different platforms? What configuration to use? Client-server (who is the client who the server?), equal partners, mixed configurations. A library was developed in SuperCollider to enable the packaging of algorithms into modules with automatic generation of GUI from specifications, and the saving of configurations of modules into session files as scripts in SuperCollider code. The application of this library as a framework for both driving graphic synthesis in Processing and receiving control data from it resulted in an environment for experimentation that is also being used successfully in teaching interactive audiovisual media.

Keywords

kinetic art, audiovisual installations, python, SuperCollider, Processing, algorithmic art, tools for multi-platform development

1.INTRODUCTION

1.1 Combining Tools to Span Different Media

The integration of different media both technically and aesthetically is one of the main challenges in art. This is especially true in art forms that involve different modes of expression and sensing such as sound, still or moving image, still or moving sculpture, text, etc. on equal terms. Digital technology presents provides new and powerful tools for addressing this challenge. However, the tools and development environments available are rarely if ever capable of spanning several media with equal ability. Most tools are specialized in one medium, or are generic programing environments that must be extended through libraries or plug-ins to work with specific media. Moreover, if working with very specific and experimental technologies such as particular types of sensors or actuators, web-based environments etc. it is hardly possible to integrate all aspects of the work in one programming tool. Thus, the ability to combine several different tools or environments becomes an important asset, if not a necessary condition, for integrating different media in works that address several modes of expression.

1.2 OSC and Communication between Applications

With the appearance of the Open Sound Control standard (OSC) [1] many applications have become able to communicate with each other. OSC has the advantage of being medium-neutral and easily configurable to meet the needs of each application independently of its specific internal mode of communication and control. Thus, OSC is now often used to connect an application to input devices as well as output and actuator devices. Less common however is the interconnection of several applications of different types. Even though this way of working is becoming popular, it has hardly been treatment as a research topic by itself. The present paper addresses precisely this issue. It is based on work done on three parallel tracks: Development of a general framework for resource management in SuperCollider, called "Lilt", the application of this framework to connect SuperCollider as a sound synthesis engine with Python to provide sound for Jean-Pierre Hébert's art projects with sand and finally educational of this framework in teaching the programming of interactive audiovisual applications.

1.3 Initial Work: Python and SuperCollider in the "Sand" Project

The present paper reports on work that started in 2004 as an experiment to add sound to a series of kinetic installations by Jean-Pierre Hébert, which draw figures on sand by means of a ball moved on a flat surface by a magnet. The magnet is controlled by a program written in Python which calculates the trajectory as a sequence of line segments of specified length and direction. The objective was to derive the sound synthesis parameters in real-time based on the data representing the position and trajectory of the ball on the sand. The data were sent from Python to SuperCollider via OSC. To facilitate development and enable experimentation with different ways of matching sound physical to movement, we developed an instrument-orchestra-score model. While such a paradigm is known from sound synthesis environments such as Csound, the present implementation differs from it in fundamental ways because it requires real-time synthesis (or "inference") of the score from the parameters of the ball movement.

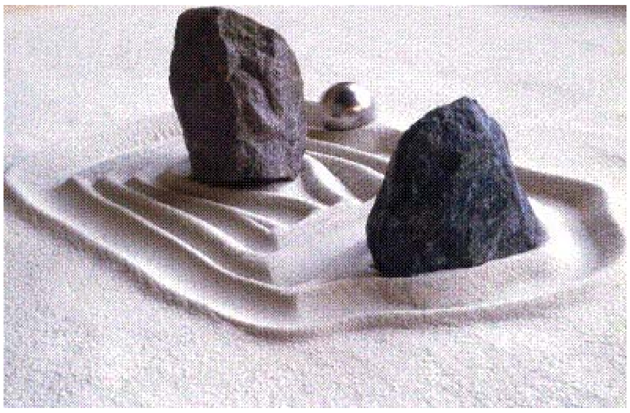


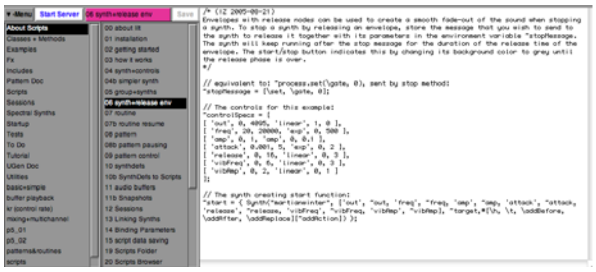
Figure 1: Example of a Sand Piece by Jean-Pierre Hébert

2. CONFIGURABILITY IN OPEN DEVELOPMENT ENVIRONMENTS

A basic motivation for the development of the library was the need to organize code so as to maximize reuse while not limiting the access of the programmer to all aspects of the system. The objective of the library was therefore not to require of the programmer to master and use the API of the given tool exclusively, but rather to offer the option of wrapping any code in a construct that provides essential features of control and interconnection. The fundamental concept that was born to address this need was that of a "Script" as a unit of code with uniform but configurable features.

2.1 The "Script" Concept

The script concept was born out of the need to create and manage a library of code snippets that realize ideas in SuperCollider. By providing a uniform interface for starting, stopping, controlling and interconnecting scripts.



```
/* (IZ 2005-08-21)
Envelopes with release nodes can be used to create a smooth fade-out of the sound when stopping
a synth. To stop a synth by releasing an envelope, store the message that you wish to send to
the synth to release it together with its parameters in the environment variable ~stopMessage.
The synth will keep running after the stop message for the duration of the release time of the
envelope. The start/stop button indicates this by changing its background color to grey until
the release phase is over.
*/

// equivalent to: ~process.set(\gate, 0), sent by stop method:
~stopMessage = [\set, \gate, 0];

// The controls for this example:
~controlSpecs = [
[ 'out', 0, 4095, 'linear', 1, 0 ],
[ 'freq', 20, 20000, 'exp', 0, 500 ],
[ 'amp', 0, 1, 'amp', 0, 0.1 ],
[ 'attack', 0.001, 5, 'exp', 0, 2 ],
[ 'release', 0, 16, 'linear', 0, 3 ],
[ 'vibFreq', 0, 6, 'linear', 0, 3 ],
[ 'vibAmp', 0, 2, 'linear', 0, 1 ]
];

// The synth creating start function:
~start = { Synth("martianwinter", ['out', ~out, 'freq', ~freq, 'amp', ~amp, 'attack', ~attack,
'release', ~release, 'vibFreq', ~vibFreq, 'vibAmp', ~vibAmp], ~target,*[\h, \t, \addBefore,
\addAfter, \addReplace]["addAction"]) };
```

Figure 2: The Script Browser

2.2 GUI Generation

One further feature of the script concept is the ability to create its own GUI for control based on a list of simple specifications that determine the names and ranges of parameters. To provide maximum flexibility, it is possible to override the default action of the GUI element that is generated by the script by a user-defined function.

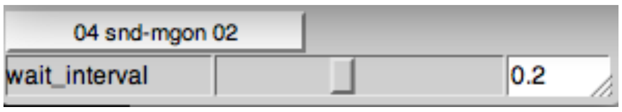


Figure 3: Basic GUI example

2.3 Connecting Scripts

Connecting Scripts refers to making one script read one input from the output of another script. For example, a script that contains a synth f that adds reverberation may read its audio input from another synth s that produces an audio output. Thus the reverberation effect of script *f* is added to the signal produced by the audio output of synth *s*. A single script may at the same time receive input from several other scripts, on one or several different of its inputs. Similarly, a single script may send its output to one or more other scripts. In most cases a script will have several inputs but only one output. The inputs of a script that runs one single synth are the synth's inputs while the output of that script is the synth's output.

Implementing the dynamic interconnection of scripts proved to be a major task. A number of constraints and conditions at different levels must be met: Synths have to be able to start and stop independently of each other, be placed in the right order of computation in the synth graph of the server, and employ the right configuration of busses for writing and reading signals. Automatically computing the right configuration of busses was perhaps the most complicated part of the work. As shown in figure 3, to enable two sources (w1 and w2 to write to two effect processes r1 and r2 where w1 writes only to r1 and w2 writes both to r1 and to r2, it is necessary to copy the output signal of w2 to the bus that reads the separate output of w1.

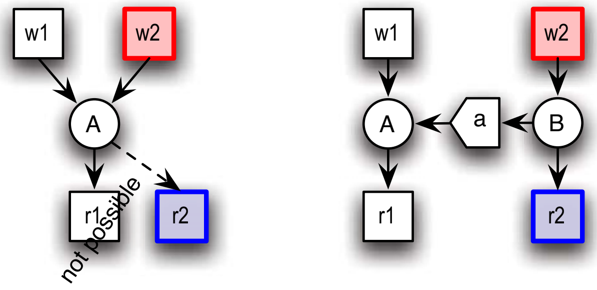


Figure 3: Signal Copying For n-to-n Configurations

An algorithm was devised that can compute the necessary bus and copying synth structures for any configuration of synth interconnections dynamically and realize it even while the synths are running. Figure 4 shows one of several cases that where analyzed in the process of developing the algorithm.

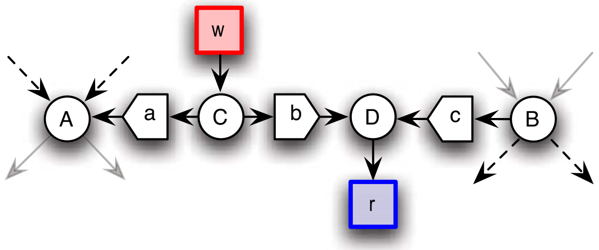


Figure 4: Generalized n-to-n Configuration Case

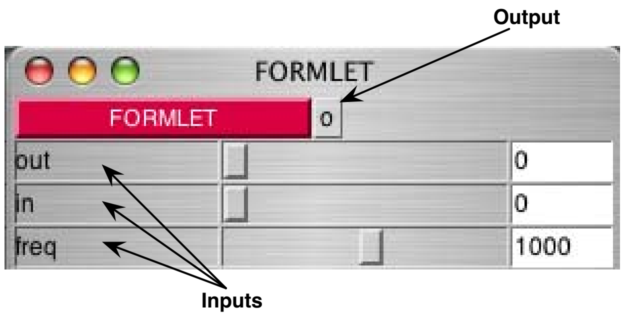


Figure 5: Inputs and Outputs in a Script GUI

2.3.1 Extensions of the Interconnection Scheme

Interconnections are not limited to audio signals, but can also be created for control signals. Additionally, there is a similar scheme for linking scripts so that they can exchange messages or function calls. This is implemented by attaching editable pieces of code, called "snippets" to scripts, which can be used to further control or automate the script's behavior.

3. RESOURCE MANAGEMENT

A characteristic difference of experimental and programmable development environments to commercial tools for image or sound processing is the relative lack of management facilities of the former. Applications such as FinalCut Pro, DVD Studio, Logic Audio, Cubase etc. use their own file formats for saving "project data" which include settings such as the paths of audio files used, processing data on the files etc. One of the objectives of the present work is to provide such resource management facilities to SuperCollider. The usefulness of such facilities is easy to demonstrate: When experimenting with several scripts that require synthesis algorithms, buffers, and bus interconnections it is convenient to be able to save the configuration of scripts, buffers, synthesis algorithms and interconnections onto file per mouse-click. This is implemented in Lilt by the concept of a Session. A session saves all the above data as a Script that can recreate the sessions elements. The Script is generated in SuperCollider code and can therefore be inspected by the user.

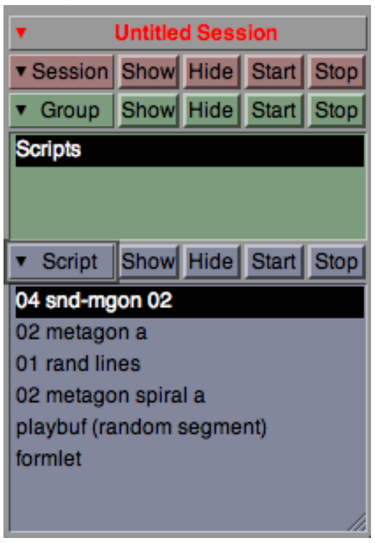


Figure 6: The Sessions Pane

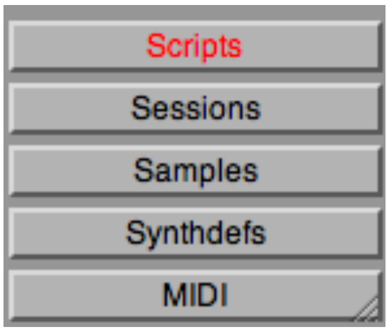


Figure 7: The Resource Pane Window

4. APPLICATION EXAMPLE: AN AUDIOVISUAL SEQUENCE

The tools described above are currently being evaluated for application in mixed media for artistic production and for education. Figure 6. shows the results of work done by two students, Alexandros Synodinos and Christos Mousas, at the Department of Audiovisual Arts at the Ionian University as part of 4th year undergraduate coursework. These students had no experience in programming at all.

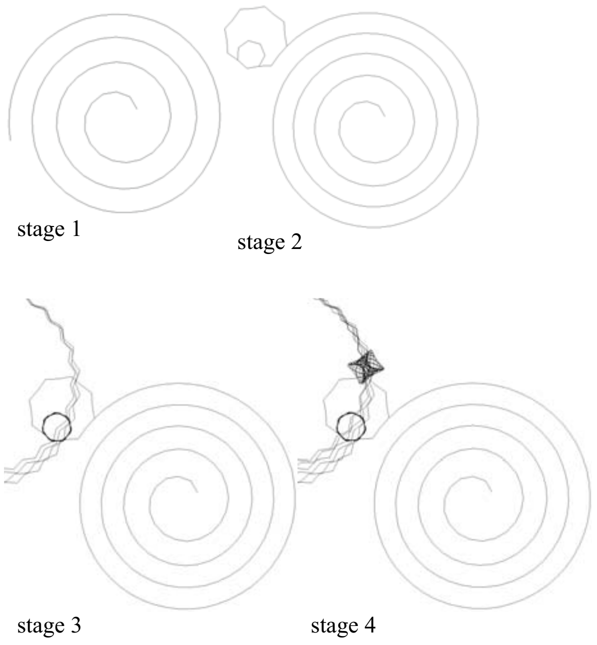
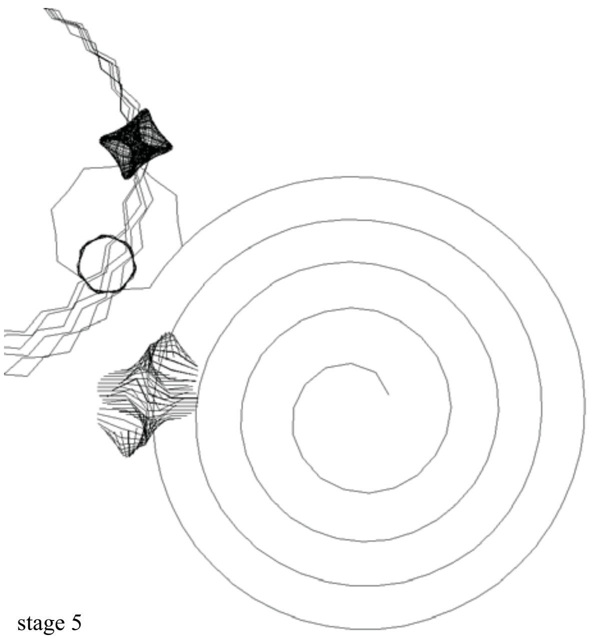


Figure 8a: Initial Sections of an Algorithmic Audiovisual Piece


stage 5

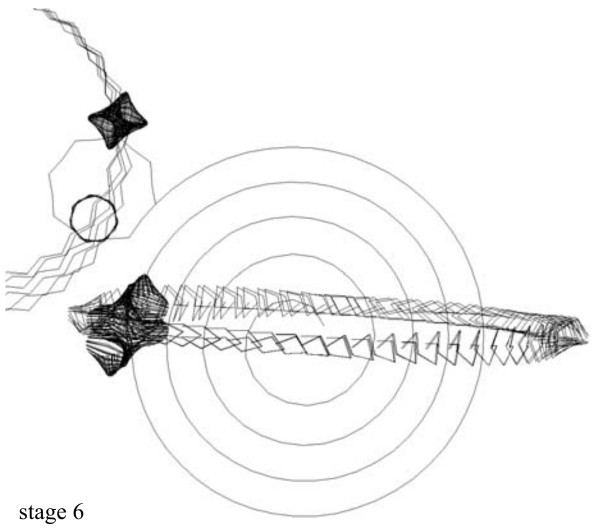

stage 6

Figure 8b: Further Stages of an Algorithmic Audiovisual Piece

The examples of Figures 8a and 8b. show several phases in the unfolding of an algorithmically composed audiovisual piece running on Processing and SuperCollider. It is visible how the students created a work with several distinct sections, starting from a given initial spiral and adding their own variations in increasing freedom. The starting point was an example provided by Jean-Pierre Hébert. This was first modified radically to reduce to the basic functioning principles. Then an interface to SuperCollider was provided using the Lilt library. Two versions were prepared: In the first one, sound synthesis on SuperCollider is driven from Processing. Conversely, in the second one, SuperCollider drives graphic synthesis on Processing. The second approach has the advantage that timing can be controlled accurately and independently from the frame rate of the draw function in Processing.

5. CONCLUSION

In this paper, we presented a framework for mixed-media interactive installations that run distributed on the three development environments SuperCollider, Python and Processing. The central part of the framework is the Lilt library written in SuperCollider, which enables the modularization and re-use of code, the easy configuration and interconnection of modules, and the saving of configurations in SuperCollider code as scripts. We showed applications that used this environment both in an artistic and in an educational setting. While the initial stages of work on this project were hard, because the design solutions were not yet mature, more recent results are encouraging. Besides the undergraduate work shown here, there exist also several graduate projects that employ Lilt for multimedia work in connection with Max/MSP and Jitter as well as vvvv (see http//vvvv.org). Certainly, the graphic elements shown in the present example are simple, and remind one of early phases in the development of the Logo environment for programmable graphics [3]. However, there is a big difference here, in that both timing and sound are involved, and that it is possible to connect further independent tools to the framework via OSC. The advantage of the present approach is that it can support the combination of software specialized in different domains, thereby helping to exploit the full potential of these applications in work that involves several different media.

6. ACKNOWLEDGMENTS

Thanks are due to 4th year undergaduate students Alexandros Synodinos and Christos Mousas for providing the visual examples in this paper.

7.REFERENCES

[1] Wright, M. and Freed, A. Open Sound Control: A New Protocol for Communicating with Sound Synthesizers. *Proceedings of the 1997 International Computer Music Conference*, Thessaloniki, Hellas (Greece), 1997, 101-104.

[2] Alvaro, J. Miranda, E. and Barros, B. EV Ontology: Multilevel Knowledge Representation and Programming, *Proceedings of the 10th Brazilian Symposium on Computer Music (SBCM)*, Belo Horizonte (Brazil) 2005.

[3] Papert, S. *Mindstorms: Children, Computers, and Powerful Ideas.* Basic Books, N.Y. 1980.