INTERACTIVE AUDIO SIGNAL SCRIPTING

Victor Lazzarini
An Grúpa Theicneolaíocht Fuaim agus Céil Dhigitigh
National University of Ireland,
Maynooth, Ireland
Victor.Lazzarini@nuim.ie

ABSTRACT
This article discusses interactive audio scripting using a Python language module, PySndObj, based on the Sound Object (SndObj) Library. This module allows for advanced music and audio scripting and provides support for fast application development and on-the-fly synthesis programming. The article introduces the main concepts involved in audio programming with the library and provides an overview of the PySndObj module with some examples. The article concludes with a discussion of interactive audio scripting with reference to the system presented here.

1 INTRODUCTION
Interactive coding of audio applications is probably one of the most exciting developments in Computer Music in recent years. The refinement of synthesis systems such as SuperCollider 3 [8] and Csound 5 [4] have allowed composers and performers to transform the computer into a powerful realtime instrument.

One interesting approach is that of harnessing the capabilities of general purpose scripting languages, such as TclTk [13] or Python [12], by adding audio processing capabilities to them. The system discussed in this article is based on this principle. A complete audio system, written in C++, is provided as a module to the Python language. A simple and transparent interface is provided to create sound processing objects and the programming language provides a very flexible environment for interactive audio scripting.

This article is organised as follows: first the basis for the audio processing module is introduced, followed by a discussion of audio programming in Python, with some examples. The details of interactive audio scripting are then explored, in relation to the present system.

2 THE ENGINE
The Sound Object (SndObj) Library [5] is an object-oriented [1] audio processing library that can provide an audio engine for interactive scripting of audio processes. The library is, in its basic form, a collection of classes for synthesis and processing of sound, inspired by the example set by the MUSIC N family of programs [3]. The library can be used for, among other things, fast application development, containing over 100 classes, supporting most types of synthesis and processing, from classic techniques to physical models, granular synthesis and spectral processing. While this article will explore audio scripting with its Python module, PySndObj, the library is also available in C++ [11], Java [2] and Common Lisp [10].

A SndObj (pronounced ‘Sound Object’) is a programming unit that can generate signals with audio or control characteristics. It has a number of basic attributes, such as an output vector, a sampling rate, a vectorsize and an input connection (which points to another SndObj). Other attributes will depend on the specialisation of this basic model.

SndObjs contain their own output signal. So, at a given time, if we want to obtain the signal it generates, we can probe its output vector. This is a basic characteristic of SndObjs: signals are internal, as opposed to existing in external buffers or busses. The basic operation that a SndObj performs is to produce an output signal. This is done by invoking the (overridable) SndObj::DoProcess() method, generally in a processing loop. For this, users can also avail of the services of the SndThread class and its derivatives, which provide processing thread management.

Since signals are internal, SndObjs do not have direct signal inputs. Instead, they will have input connections to other SndObjs. For connecting and disconnecting, as well as scalar parameter setting, a message-passing interface is defined, with the SndObj::Set() and SndObj::Connect() methods.

The Sound Object Library provides classes for time and frequency-domain processing. These two types SndObjs are designed to fit in together very snuggly in a processing chain. The library offers streaming spectral processing, which can be provide a flexible way of developing powerful realtime sound manipulation programs.

Signal input and output is handled by SndIOs (‘sound ios’), which are objects that can write and read to files, memory, devices, etc. Their main performing methods are SndIO::Read() and SndIO::Write(). When invoked, these will read or write a vectorsize full of samples from/to their source/destination, respectively. SndIOs can handle multichannel streams, so their output vector actually contains frames of samples (in interleaved format). For signal input, SndIO channels can be accessed via separate SndObjs, using the SndIn class. For signal output, SndObjs are connected directly to a SndIO.

As certain SndObjs will use tabulated functions, a special type exists, a Table object. These are very simple objects whose most important attribute is their actual tabulated function, which is created at construction time. Tables can be updated at any time, by changing some of their parameters and invoking Table::MakeTable().
3 AUDIO SCRIPTING IN PYTHON

PySndObj[7] is a Python module that provides a scripting interface to the library for fast application development, prototyping of applications and general on-the-fly, interactive, synthesis and processing. In a Python script, or at the interpreter,

```python
import sndobj
or
from sndobj import *
```

allows access to all SndObj library classes available to your platform, plus some extra utility classes for array support.

2.1 Python SndObjs

Using SndObjs in Python is very straightforward and transparent. The origination library design has been vindicated by this simplicity and elegance, which fit like a glove with the overall goals of Python programming. For instance, hooking a table to a lookup oscillator looks like this:

```python
tab = HarmTable()
osc = Oscili(tab, 440, 16000)
```

The same principle works for SndObj and SndIO connections, so if we set up a RT output object, we can connect our oscillator to it:

```python
outp = SndRTIO(1)
outp.SetOutput(1, osc)
```

This works between SndObjs as we would expect, if we want to, say, connect a modulator to our oscillator:

```python
mod = Oscili(tab, 2, 44)
osc.SetFreq(440, mod)
```

3.1 Running SndObjs

In order to get audio processing out of a SndObj, it is necessary to invoke its DoProcess() method. For a continuous output stream, continued calls to this method are required, so we need to set up a loop, where this can happen. In addition, any SndIOs in the chain have also to call their Read() or Write() methods (for input or output respectively).

```python
# 10 seconds of audio output
timecount = 0
end = 10 * osc.GetSr()
vecsize = osc.GetVectorSize()
while(timecount < end):
    mod.DoProcess()
    osc.DoProcess()
    outp.Write()
    timecount += vecsize
```

The simplest way to get SndObjs producing audio is to use a SndThread object to control the synthesis. This will take care of setting up a processing loop and call all the required methods. We start by setting the object up:

```python
thread = SndThread()
thread.AddObj(mod)
thread.AddObj(osc)
thread.AddObj(outp, SNDIO_OUT)
```

Then we can turn on the processing to get some audio out:

```python
thread.ProcOn()
```

When we are done with it, we can turn it off:

```python
thread.ProcOff()
```

In addition to SndThread, it is also possible to use SndRTThread, which has default realtime objects for input and output that can be connected to. In this case the user only needs to set up his/her SndObj chain and add this to the thread object.

Any asynchronous Python code can also be called, by setting up a process callback, that will be invoked once every processing period. This happens after the input signal(s) has been obtained, but before any processing by SndObjs.

For instance, if we have a method:

```python
def callb(data):
    ... # callback code
```

This sets the callback, data is any data object to be passed to the callback:

```python
thread.SetProcessCallback(callb, data)
```

The callback can be used for things like updating a display, changing parameters cyclically, polling for control input, etc..

3.2 Support for arrays

In order to facilitate programming, some utility classes have been added for int, float and double arrays, named, respectively: intArray, floatArray and doubleArray. These classes can be used as follows

```python
# create an array of two items
f = floatArray(2)
# array objects can be manipulated
# by index as in C
f[0] = 2.5
```

In addition, a special type of array is also available, the sndobjArray, which holds SndObjs (internally C++ SndObj pointers). Objects of this type can be used similarly to the above array:

```python
objs = sndobjArray(2)
objs[0] = mod
objs[1] = osc
```

However, when these are used as SndObj pointer arrays, they will need to be cast as that. Hopefully the class has a handy method for doing just that:

```python
objp = objs.cast()
```

These can be used with objects that take arrays of SndObjs as input, such as SndThread:
in which case we are setting up a thread with two SndObjs (which is similar to the example above). Other objects that take SndObj arrays are for instance SndIO-derived objects and Mixer SndObjs. But remember, SndObj arrays are not sndobjArray objects, but can be retrieved using sndobjArray::cast().

4 EXAMPLES
Two examples are shown here: a simple audio processing script and a minimal signal analysis and display software.

4.1 Simple echo using a comb filter
This example demonstrates realtime audio IO and some delayline processing, running for 60 seconds. Scripts such as this one can be run interactively from the Python shell, as discussed in the next section.

```python
from sndobj import *
import time

# SndRTThread with 2 channels
t = SndRTThread(2)

# Echo objects take input from
# SndRTThread inputs
comb_left = comb(0.48, 0.6, \n    t.GetInput(1))
comb_right = Comb(0.52, 0.6, \n    t.GetInput(1))

# We now add the echo objects to
# the output channels
t.AddOutput(1, comb_left)
t.AddOutput(2, comb_right)

# This connects input to output
t.Direct(1)
t.Direct(2)

# turn on processing for 60 secs
t.ProcOn()
time.sleep(60.0)
t.ProcOff()
```

4.2 Spectral display with Tkinter
This example shows the use of Tkinter, the default GUI module in Python, together with a processing callback. The graphics code, in the 'display.py' module, is not included here. The spectral display is shown on fig.1:

```python
from sndobj import *
from Tkinter import *
import array
import display

# window size, refresh interval, norm
# highest frequency of display
window_size = 300
time_interval = 0.1
norm = 32768.0/10
highest = 10000

def callb(data):
    re = array.array('f')
    im = array.array('f')
    fft = data[0]
    disp = data[1]
    sr = fft.GetSr()
    end = int(fft.GetFFTSize() * \n        2*highest/sr)
    for i in range(0, end, 2):
        re.append(fft.Output(1)/norm)
        im.append(fft.Output(1 + i)/norm)
    if not disp.error:
        disp.draw(re, im, \n            (sr/fft.GetHopSize())* \n            time_interval)

# SndObj chain and process thread
win = HammingTable(1024, 0.5)
thread = SndThread()
inp = SndRTIO(1, SND_INPUT)
sig = SndIn(inp, 1)
fft = FFT(win, sig);

# display object
disp = display.Spectrum(Tk(), \n    window_size, thread.ProcOff, \n    "white", "black")
dat = (fft, disp)

# thread set-up
thread.SetProcessCallback(callb, dat)
thread.AddObj(inp, SNDIO_IN)
thread.AddObj(sig)
thread.AddObj(fft)
thread.ProcOn()

# run the display
disp.mainloop()
```

Figure 1. The spectral display

5 INTERACTIVE SCRIPTING
Python provides a straightforward environment for interactive scripting, as the interpreter can be run from a variety of environments. The most universal way is to invoke the Python shell from the command-line.
Facilities for command recall and completion are not implemented on all platforms, so this is probably not the fastest environment for code typing. Several other shells exist providing various facilities for interaction. Another option is to run the emacs text editor in its Python mode, and use the interactive text selection interpretation, where parts of an existing script can be run at different times.

The PySndObj module itself is very adaptable to interactive operation. Objects can be created, connected, disconnected, modified, etc with single lines of code. The first of the examples shown above can, for instance, be run interactively, and its processing parameters can be modified on-the-fly, either by single lines interactively interpreted or programmatically, by invoking chunks of code that would act on them. Since all of these actions are only limited by the capabilities of the Python language, there is plenty of scope for various ways of interaction.

One very interesting variant on these ideas is the possibility of distributed processing of SndObjs. Using the socket module, it is possible to run various parallel processes on different machines, either from a central controller, or in a more complex mesh of interaction. All we require is to have a minimal server script running on the target machines. With socket connections, strings containing code snippets can be sent to these machines, where they are interpreted, resulting in some signal processing operation. This was demonstrated in [6], under TclCsound [9], but the same principle (in fact with an even simpler server script) can be applied to Python and PySndObj. The server interpreter code boils down to the following lines (this is a method of a ‘sndobj-server’ class that manages all the socket connections):

```python
def interpret(self):
    while not self.end:
        r,w,x = select.select(self.fd, [], [], 0)
        for i in r:
            try:
                socks = socket.fromfd(i, socket.AF_INET,
                                      socket.SOCK_STREAM)
                data = socks.recv(1024)
                exec data
            except:
                print "exception in: " + data
```

Around the above code, we just need to place a method for monitoring and managing network connections. On the client side, all we need to do is to connect to one of these servers and send Python code (as text strings) to them. An interface can be built to make the coding appear the same as if it had been done at the local interpreter.

6 CONCLUSION

PySndObj provides a good support for interactive audio programming in Python. The simplicity of the language, allied to the modularity and comprehensiveness of the library proves to be a powerful combination. All of the SndObj tools are Free software, GPL licensed, available at [http://sndobj.sf.net](http://sndobj.sf.net). Developers are also encouraged to join the project and can do so by contacting the author at his e-mail address.

7 REFERENCES