ScPy: A SuperCollider Extension for Performing Numerical Computation via Embedded Python

Noah Weninger Abram Hindle

August 26, 2016

Abstract

SuperCollider [10], a language for sound synthesis and algorithmic composition of audio, supports a wide range of synthesis, effect and analysis algorithms. However, available operations are limited to those implemented explicitly as Unit Generators (UGens). Since UGens are written in C/C++ and loaded as plugins during the SuperCollider server boot process, it is impossible to live code UGens, which limits the user to creating sound as a composition of existing UGens during a performance. Many of the vector operations required for efficiently creating complex audio effects are notably missing or tedious to use. To overcome this, we present ScPy, a UGen which embeds Python within SuperCollider to enable the use of the optimized vector operations provided by the NumPy library.

1 Introduction

Although a number of open-source projects which interface other languages with SuperCollider exist [5–7], they are mostly SuperCollider clients, which have features to control synths and send other automation messages. One notable exception is OctaveSC [3], which embeds GNU Octave within SuperCollider. However, it is designed for performing operations on control rate data arrays and suffers from performance issues which prevent it from being used on audio rate data.

One of the goals of this project is to enable more flexible experimentation with FFT and phase vocoder based operations. SuperCollider contains a total of 32 built in phase vocoder operations, which provide the ability to produce many simple effects. There are additionally some extra operations available as user created extensions. However, many conceivable effects are impossible, including basic phase vocoder effects such as pitch shifting. Some examples of novel effects can be found in Section 3.

ScPy enables users to overcome these limitations. By embedding the Python programming language within SuperCollider, it is possible to do numerical processing operations which would be too slow to perform in real time with SuperCollider alone. Through access to the NumPy library, previously difficult

or impossible phase vocoder operations become trivial. Although our usage of this library for testing purposes has mostly focused on FFT based operations, support exists for passing arbitrary data buffers to Python, which can be used to process numerical data for many other purposes as well.

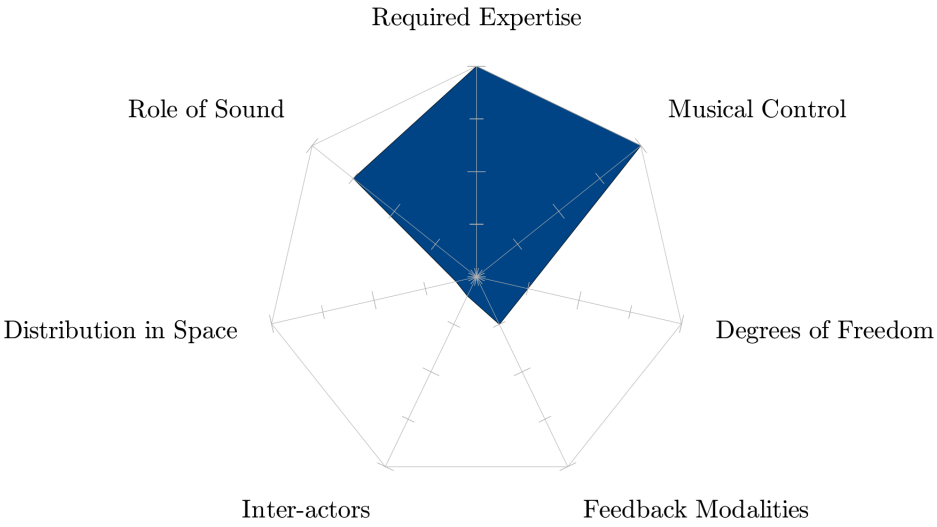
ScPy is developed and maintained as an open-source project under the GPLv3 license. The latest code and documentation is available on GitHub[1].

2 Methodology

During the initial planning stage of the project, we wanted to essentially provide a set of fundamental operations on spectral data which could be composed together to perform any conceivable effect. It was also a requirement that these operations would be syntactically concise to enable easy and fast experimentation. Ideally, the system would be able to handle complex effects in real-time.

ScPy is unlike many New Instruments for Musical Expression (NIMEs) because it does not have the properties of any traditional instrument. However, it can still be considered a NIME because it ultimately is a tool to enhance musical expression, just with an unconventional method of interaction. Dimension space analysis, as explored by Birnbaum et al. [1], can facilitate the comparison of use cases for NIMEs.

Figure 1: NIME dimension space analysis for ScPy.



Although designing a domain-specific language (DSL) [8] for this task was briefly considered, we eventually decided to use Python for a number of rea-

[1] `https://github.com/nwoeanhinnogaehr/ScPy`

sons. First, Python includes an excellent C API which makes embedding it within other languages trivial in comparison to many other options. Second, Python's syntax is easily readable and concise — it clearly adheres to our goals. Finally, Python has a massive number of community built libraries for performant scientific and numerical computing. With a few small exceptions, which are clearly specific to spectral data (such as the phase vocoder), these libraries contained every one of the fundamental operations we could think of.

It may seem as though embedding an interpreted language within an interpreted language could offer no performance advantage. There is certainly some performance hit when switching languages and transferring data. However, the biggest advantage comes not strictly from performance but from the diversity of operations which become available with highly optimized implementations. NumPy is used internally by ScPy for handling data arrays, so the entire NumPy library of mathematical functions comes at no cost. Any other Python library can easily be imported as well. Limited experiments have been done with the SciPy library. It would also be possible to use Theano to do processing on the GPU for extra performance. Implementing an equivalent to these popular Python libraries in pure SuperCollider is certainly possible, but it would require a very large amount of work and would see no adoption outside of the audio computing community.

3 Usage

Use of ScPy is through the classes `Py` and `PyOnce`. `Py` is used for processing real-time data, and `PyOnce` is used for doing initialization or other one time processing. Both have one required argument, a block of Python code. A map of variable names may optionally be provided to bind SuperCollider variables to Python variables. Additionally, there is an optional `DoneAction` argument which specifies an action to occur in SuperCollider after the Python code has finished executing.

```
(PyOnce("
    # comments in Python have different syntax!
    print('Hello from Python!')
    print('x is', x)
", (x: 62)))
```

Listing 1: Hello world with ScPy.

```
(s.waitForBoot {
    var buf = { Buffer.alloc(s, 512) }.dup;
    var hop = 1/4;

    s.freeAll; // stop previous versions

    PyOnce("
        def fn(x):
            return x
    ", (hop:hop));

    {
        var in = AudioIn.ar([1, 2]);
        var x = FFT(buf.collect(_.bufnum), in, hop);
        Py("
            out(x, fn(array(x)))
        ", (x:x, time:Sweep.kr));
        Out.ar(0, IFFT(x));
    }.play(s);
})
```

Listing 2: SuperCollider boilerplate for no-op FFT effect with ScPy.

When Listing 2 is run, an anonymous synth is created which will take in audio from an external source, perform a STFT, process the spectral data with ScPy, perform an inverse STFT, then finally output the resulting audio. Lines 8–9 contain Python code defining the processing function where operations are to be inserted. Further examples will replace only those lines. Line 16 contains the Python code to convert the spectrum into a NumPy array, process it, then write back the processed version.

```
pv = PhaseVocoder(hop)

def fn(x):
    x = pv.forward(x)
    idx = indices(x.shape)[1]
    x = pv.shift(x, lambda freq:
        freq * (0.8 + mod(-time + 0.1*idx, 10)*0.045))
    x = pv.backward(x)
    return x
```

Listing 3: A novel effect to transform any sound into a falling Shepard tone. This example makes use of the phase vocoder `shift` operation, which applies a function to frequency, then reorganizes the spectrum to move the new frequencies into the appropriate bins.

```
pv = PhaseVocoder(hop)

def fn(x):
    x = pv.forward(x)
    x = pv.to_bin_offset(x)
    x.imag = -x.imag
    x = pv.from_bin_offset(x)
    x = pv.backward(x)
    return x
```

Listing 4: An effect which inverts the frequency of each analysis bin across the center of the bin, creating an unusual detuning effect.

4 Implementation

ScPy is written mostly in C++, but with a small SuperCollider class library to connect the C++ back-end, and a small Python library that provides some useful operations. Due to differences between these languages, connecting them was somewhat awkward in some cases.

Since UGens may only be used as part of a synth, `PyOnce` is provided as a wrapper which hides this detail from the user and allows the execution of Python code outside of a synth. It is useful for doing initialization work. Since the current implementation executes Python code in a single global namespace, a typical usage patten is for state to be defined in a `PyOnce` block which is later accessed and modified in a `Py` block.

One issue involves the process of passing data between SuperCollider and C++. Since UGens are designed to work primarily with data streams, adding support for passing in many types of data came with a number of challenges. Essentially, all data must be serialized into an array of floats. Strings must be converted to an array of ASCII character codes, and prepended with their length. Arguments passed through to the Python code can have one of many types, and therefore type information must be encoded as well. However, just passing the type of a variable is not enough to know what we can do with it, since we also want to know if it inherits from a class we can use.

Since `FFT` objects in SuperCollider operate on a single channel only, they must be placed in `Array` objects to support multichannel audio. `Array`s can hold objects of any type, so it is natural to convert them to Python `list` objects, which have similar properties. However, this presents an interesting problem: multichannel `FFT` objects are then converted to `list`s of `ndarray` objects in Python. These are a bit more awkward to work with than basic `ndarray` objects, but can easily be converted using a call to the `array` function in Python. This can be seen in practice on line 16 of Listing 2. Although it would be possible to do this conversion automatically, it is left out for the purpose of not introducing a feature simply to work around an incomplete implementation of `FFT` objects in SuperCollider. The current conversions will continue to function as intended

if `FFT` objects are given multichannel support in the future.

Table 1: Type correspondence between SuperCollider and Python

SuperCollider type	Python type
`Float`	`float`
(subclass of) `UGen`	`ndarray` of `float`
`Buffer`	`ndarray` of `float`
`FFT`	`ndarray` of `complex`
`Array`	`list`

As shown in Listing 3 and Listing 4, ScPy includes a phase vocoder implemented in Python for performing frequency transformations. Interestingly, although all of SuperCollider's FFT operations are referred to as phase vocoder operations, there is actually no implementation of a phase vocoder to be found in the SuperCollider code base at the time of writing. It is likely that a phase vocoder was once intended to be implemented, but the 32 so-called phase vocoder effects are simply transformations to the Cartesian or polar representations of the spectrum.

5 Future Work

At the time of writing, ScPy only supports handling audio rate data via SuperCollider's `Buffer` objects. In order to make some use cases more ergonomic, ScPy could certainly be extended to support raw audio rate input. For example, this would make time domain audio data even simpler to work with than frequency domain data is currently. It would also enable SuperCollider's FFT UGen to be easily re-implemented in Python for greater flexibility.

Support for more SuperCollider types, such as events or strings, could be added to open up ScPy to more use cases.

Multi-core processing support would allow for many more sounds to be processed in real-time.

Creating a more generic extension which supports many languages would have a number of advantages. One option for implementing this would be to define a specification to communicate and share data with a generic UGen in a running SuperCollider instance.

Although extending SuperCollider was the focus of this project, similar improvements could be made to the CSound[2, 4], Chuck[9] or Faust[6] environments as well.

References

[1] David Birnbaum et al. "Towards a dimension space for musical devices". In: *Proceedings of the 2005 conference on New interfaces for musical expression*. National University of Singapore. 2005, pp. 192–195.

[2] Richard Charles Boulanger. *The Csound book: perspectives in software synthesis, sound design, signal processing, and programming*. MIT press, 2000.

[3] Thomas Hermann. *OctaveSC*. `http://www.sonification.de/projects/sc3/index.shtml`. 2006.

[4] Victor Lazzarini. "Extensions to the Csound language: from user-defined to plugin opcodes and beyond". In: *Proc. of the 3rd Linux Audio Conf.* Citeseer. 2005, pp. 13–20.

[5] Thor Magnusson. "ixi lang: a SuperCollider parasite for live coding". In: *Proceedings of International Computer Music Conference*. University of Huddersfield. 2011, pp. 503–506.

[6] Yann Orlarey, Dominique Fober, and Stéphane Letz. "Faust: an efficient functional approach to DSP programming". In: *New Computational Paradigms for Computer Music* 290 (2009).

[7] *Systems interfacing with SC*. `http://supercollider.sourceforge.net/wiki/index.php/Systems_interfacing_with_SC`. July 2011.

[8] Arie Van Deursen, Paul Klint, and Joost Visser. "Domain-Specific Languages: An Annotated Bibliography." In: *Sigplan Notices* 35.6 (2000), pp. 26–36.

[9] Ge Wang. *The chuck audio programming language. a strongly-timed and on-the-fly environ/mentality*. Princeton University, 2008.

[10] Scott Wilson, David Cottle, and Nick Collins. *The SuperCollider Book*. The MIT Press, 2011.