Abstract

SuperCollider [McCartney, 1996] is a programming language and environment specialized for real time audio synthesis. SuperCollider version 2.0 (SC2) is a complete rewrite of the system to make it more expressive, more efficient, and easier to use. This required rethinking the implementation in light of the experience of the first version. It is my opinion that the new version has benefitted significantly from this rethink. It is not simply version 1.0 with more features. This paper will give a brief overall description of the system followed by descriptions of a few specific features.

1 The Language

SuperCollider is now a fully object-oriented language very much like Smalltalk [Goldberg, 1989]. Many of the class names and method names are the same as those in Smalltalk. The class library currently consists of over 250 classes, to which users can add their own. There are Collection classes such as Arrays, Dictionaries, Sets, SortedLists, Bags, etc. There are a number of differences from Smalltalk that facilitate real time operation and computer music applications. Among these are: a real time garbage collector, a method lookup table so that method lookup is as fast as for C++ while still retaining the dynamism of Smalltalk style methods [Driesen, 1995], positional arguments, variable length argument lists like LISP's #restor C's varargs, default values for unsupplied arguments, initial constant values for local variables, full lexical closure semantics as in Scheme, ability to run at interrupt level.

Why invent a new language and not use an existing language?

Computer music composition is a specification problem. Both sound synthesis and the composition of sounds are complex problems and demand a language which is highly expressive in order to deal with that complexity. Real time signal processing is a problem demanding an efficient implementation with bounded time operations. There was no language combining the features I wanted and needed for doing digital music synthesis. There was no other language readily available that was high level, real time and capable of running at interrupt level.

Why use a text based language rather than a graphical language?

There are at least two answers to this:

Dynamism: Most graphical synthesis environments use statically allocated unit generators where one screen icon represents one unit generator. In SuperCollider, the user can create structures which spawn events dynamically and in a nested fashion. Patches can be built dynamically and parameterized not just by floating point numbers from a static score, but by other graphs of unit generators as well. Or you can construct patches algorithmically on the fly. This kind of fluidity is not possible in a language with statically allocated unit generators.

Brevity: In SuperCollider, symmetries in a patch can be exploited by either multichannel expansion or programmatic patch building. For example, the following 5 line program generates a patch of 49 unit generators. In a graphical program this might require a significant amount of time and space to wire up. Another advantage is that the size of the patch below can be easily expanded or contracted just by changing a few constants, and it can be done on the fly, per event.
2 The Synthesis Engine

A number of improvements have been made in version 2.0 to make the synthesis engine both faster and higher quality. There are many new unit generators in version 2.0 and many of those retained from version 1.x have been improved.

The synthesis engine is a two rate system, with an audio sample rate and a sample block size which determines the control rate. Unit generators either calculate a block of samples or a single value per control period. Control rate signals are now always linearly interpolated up to audio rate whenever they are used in order to eliminate zipper noise.

There is a unit generator named Spawn which can spawn sub-events. Each spawned event can have its own control rate block size. Spawned event start times and stop times are single sample accurate.

The synthesis engine is now separate from the language virtual machine. This makes it more efficient than in version 1 where the synthesis loop was executing language code every control period.

Audio buffers such as sample files and delay lines can be allocated dynamically. Delay line unit generators do not require that their buffer be zero filled to begin with, which means there is no glitch if you decide to allocate and begin using an 8 second delay line on a moment's notice.

3 The Environment

SC2 is a fairly nice simple styled text editor which has an embedded interpreter. Language examples may be executed directly from a document by selecting them. This allows you to write pieces that are their own documentation. There are no proprietary format patch files. SuperCollider compiles text files and reads sound files as independent objects rather than being bundled together as they were in SuperCollider 1.

There is a built in help system that will open help files for any class and generate templates for any method. Since SC2 is a text editor, all of the documentation can be opened by SuperCollider directly and the examples can be executed right from the documentation.

4 Programming

Writing synthesis algorithms is more straightforward in version 2 and results in cleaner and clearer programs than in version 1.

UGen is the name of the class of unit generators. A unit generator is created by sending the 'ar' or 'kr' message to the unit generator's class object which will create instances that run at audio rate or control rate respectively. There are currently over 160 different unit generators available to work with.

SinOsc.ar(800, 0, 0.2); // create a sine oscillator at 800 Hz, phase 0, amplitude 0.2

A unit generator's signal inputs can be other unit generators, scalars, or arrays of unit generators and scalars.
In order to play a unit generator, it needs to be installed in a Synthobject. A Synth is a container for one or more unit generators that execute as a group.

```ruby
Synth.new(SinOsc.ar(800, 0, 0.2)).play;
```

or more briefly:

```ruby
Synth.play(SinOsc.ar(800, 0, 0.2));
```

You can do math operations on unit generators and the result will be another unit generator. Doing math on unit generators does not do any signal calculation itself - it builds the network of unit generators that will execute once they are played in a Synth. For example, the following code builds a network of three unit generators (sine oscillator, noise generator, adder):

```ruby
SinOsc.ar(800, 0, 0.2) + BrownNoise.ar(0.2);
```

Most unit generators have multiply and add inputs which allow a faster in place multiply-add to be performed. If these inputs are not specified then the additional operations are not performed so there is no overhead to having this feature. The example above could be written as follows by using this feature:

```ruby
SinOsc.ar(800, 0, 0.2, BrownNoise.ar(0.2));
```

## 5 Events

The Spawn unit generator spawns events and mixes them to a number of output channels. Spawn takes as an argument a function that creates a network of unit generators. Spawn calls this function for each event. Because Spawn is a unit generator that is a mix of sub-events, it makes it easy to apply effects processing to an entire mix, or to have controllers that range over a series of events such as a phrase envelope. Spawned events are spliced together with single sample accuracy and each event may have its own sample block size. Spawn takes care of all the mixing and splicing.

```ruby
( e = Env.linen(2, 5, 2, 0.02); // 9 second trapezoid envelope specification, amplitude 0.02
  Synth.play(
    // spawn random panned sine waves of random frequency each 1/2 second
    Spawn.ar(
      Pan2.ar // two channel panner
      FSinOsc.ar(2000.0.rand, EnvGen.kr(e)), // sine oscil with an envelope
      1.0.rand2) // random pan position
    ),
    2, // two output channels
    0.5) // new event each 1/2 second
)
```

Instances of Spawn can spawn sub-events which themselves contain instances of Spawn that spawn sub-sub-events and so onto any depth.

An instance of Spawn may have any number of output channels and sub-events (which themselves may have any number of channels) may be placed at a particular channel in the output. By separating the output channels one may apply different effects to different channels of a Spawn, thereby using Spawn like a mixer with different effects sends on each channel.

Spawn can be used as a building block for other higher level constructs such as score players. The two classes XFadeTexture and OverlapTexture are built upon Spawn and are used to generate continuous cross fading textures from any synthesis function.
6 External Control

External control is implemented via the ControlIn unit generator. ControlIntakes as an argument a control source which may be re-patched at any timeto any other source. Sources may be any MIDI source such as controllers, pitch bend, pressure, etc., or mouse x or y position, graphical user interface controls, a floating point variable, or anything that responds to the 'value' message with a Float such as a function or a stream or practically anything else. ControlIn has a built in one pole smoothing filter to alleviate zipper effects. The control source's domain can be mapped either linearly or exponentially to a desired range.

// Mouse control of a sine wave frequency
Synth.play( SinOsc.ar(ControlIn.kr(MouseX.new(200, 2000, 'exponential')),0, 0.4) )

or more briefly (the MouseX object knows how to wrap itself in a ControlIn object):

Synth.play( SinOsc.ar(MouseX.kr(200, 2000, 'exponential'), 0, 0.4) )

7 Graphical User Interface

SuperCollider contains a graphical user interface builder and code generator in order to implement synthesis control panels. Windows can be built by dragging and dropping controls and then generating the code to instantiate the window. In version 1 there was only one control panel window, but version 2 may have many control panels open at once and they can be instantiated by user code.

8 Conclusion

Why is SuperCollider interesting? SuperCollider combines real time performance and control with a high level programming language, dynamic patch construction, and a graphical user interface builder. Its styled text editor interface makes it ideal for demonstrating synthesis algorithms in an interactive tutorial. The ability to add ones own classes to the library makes it a good environment for exploring higher level constructs.

9 Availability

SuperCollider currently runs only on PowerMacintosh. A development version of SuperCollider 2 can be obtained from the directory ftp://www.audiosynth.com/pub/SC2/

The exact file name will depend on the version currently available. The final version will be announced on http://www.audiosynth.com

References

