

Workshop: Synthesis and Performance in SuperCollider

H. James Harkins

March 23, 2014

Contents

1	Preface	4
1.1	Preface	4
I	Introductory SC, Synthesis and Sequencing	7
2	Workshop introduction	7
2.1	Workshop introduction	7
2.2	Brief, incomplete history of audio programming	8
3	SC architecture, usage	10
3.1	SC architecture	10
3.2	Using the Integrated Development Environment	12
3.3	Preparing the environment	13
4	SC language: Beginning steps	16
4.1	Making noise	16
4.2	SC language syntax	18
4.3	Exercises: Math operations	26
5	Modular synthesis with JITLib	28
5.1	What is modular synthesis?	28
5.2	JITLib patching	30
6	Note control	32
6.1	Playing notes by envelopes	32
6.2	Timed and sustaining envelopes	37

7 Sequencing	40
7.1 Overview: Routines and Tasks	40
7.2 Time control: Clocks and scheduling	44
7.3 Sequencing of notes	46
7.4 Data streams for note information	49
II Sequencing with Patterns; Synthesis Techniques	54
8 Events and Sequencing	54
8.1 Events and synthesis control	54
8.2 Patterns and Events	58
9 Subtractive synthesis	62
9.1 Overview	62
9.2 Analog-style oscillators	64
9.3 Filters	70
10 Modulation: Low-Frequency Oscillators	73
10.1 Modulation: Low-Frequency Oscillators	73
10.2 Range mapping for modulation	75
10.3 Another envelope use: Articulation	80
11 Frequency modulation (FM) synthesis	82
11.1 John Chowning's experiment	82
11.2 Refining FM synthesis	84
11.3 Wavetable oscillators	86
12 Sample playback and manipulation	90
12.1 Loading and playing samples	90
12.2 Triggering sound file fragments	94
III Musical and External Control	103
13 Granular synthesis	103
13.1 Granular synthesis parameters	103
13.2 Usage of granular synthesis	107
14 External control	110
14.1 Basic concepts: GUI	110
14.2 Open Sound Control fundamentals	119
14.3 OSC and mobile control	121

15 Toward complex composition	126
15.1 Simple ideas	126
15.2 Composition: Representing musical information	128
15.3 Composition: Phrase structures	133
16 Considerations for group composition	136
16.1 Group composition: Technical issues	136
16.2 Group composition: Creative issues	139
IV Effects and Mixing	143
17 Effects and mixing	143
17.1 Mechanics: Applying effects	143
17.2 Shared effects	147
17.3 Common effects: Chorus	151
17.4 Common effects: Distortion	154
17.5 Common effects: EQ	158
17.6 Common effects: Reverb	159
V Canonical Style	160
18 Canonical-style synthesis	160
18.1 Converting to canonical style	160
18.2 SynthDefs	161
18.3 Canonical style: Additional topics	164
18.4 But really, what to do next?	166
VI Appendix: Additional Topics	167
19 Modal synthesis	167
19.1 Ringing filters	167
19.2 Banks of ringing filters	169
19.3 Formant synthesis	171
19.4 Karplus-Strong	177
20 Programming concepts for composition	183
20.1 Composition: Data structures	183
20.2 Composition: Control structures	192
21 Synchronizing interfaces: Model-View-Controller	197
21.1 Model-View-Controller object design	197

VII Indices	206
22 Glossaries	206
22.1 Concepts	206
22.2 UGens	215
22.3 Other classes	219
22.4 Methods	225
23 List of Code Examples	232

1. Preface

1.1. Preface

Project history

In autumn 2013, 张睿博 (Zhang Ruibo, Shenyang Conservatory and head of CHEARS, the China ElectroAcoustic Resource Survey) approached me to present a week-long introductory SuperCollider workshop in April 2014. I wrote this material in the first couple of months of 2014.

Overview

This document includes all the presentation slides from the workshop, plus further explanation in text, an Appendix with additional topics (interesting, but too much for one week!), and an index and glossary of terms. The material serves two purposes: first, to guide the workshop sessions (which don't need to follow the slides exactly), and second, to provide documentation which CHEARS may translate into Chinese.

The workshop aims to simplify individual units of code by imitating *modular* synthesis in code. This approach avoids several of the trickier challenges of learning SuperCollider, related to code structure and server architecture. It is, however, a new approach; “canonical” SuperCollider style is different in some important ways.

Why not simply teach the canonical style? When approaching any programming language, small code blocks are easier to understand, while larger ones, even ten lines, may be scary at first. Canonical style binds many synthesizer components into one self-contained unit called a `SynthDef`; dividing it into more digestible pieces requires the new user to understand details of the server that are likely to be confusing at first. The modular style presented here allows each module to handle one small piece of synthesis *and* relieves the new user from the burden of understanding advanced concepts too early.

This modular style depends on some new *extensions* to the language. Installation instructions are in Part I. The specific extensions are:

- Improved handling of modules that represent constant numbers.
- A modifier, `\psSet`, to act on many modules at once using a central code interface.

Some of these may be incorporated in future SuperCollider versions. At that time, the workshop extensions package will have to change.

Typographical conventions

Code examples appear in a monospaced sans-serif font, Inconsolata:¹ `Server.local.boot`.

Keyboard shortcuts are boxed: `Ctrl-Return`.

Numbered code examples are collected into `scd` files. Each Part has two collections: one with all examples, and the other with only selected examples. I strongly recommend using the selected examples only. You will learn more by typing the examples yourself, instead of passively executing them.

The document is written using *org-mode* 8.2.3² running in Emacs 23, and typeset using \LaTeX (*xelatex* from TeXLive 2012³) with *Beamer* extensions.⁴

Copyleft and licensing

This document and all associated materials are released under the Creative Commons Attribution-ShareAlike 4.0 license.⁵ You may use these materials as the basis of a new workshop or course, if you:

- Give credit to me as the initial author (Attribution);
- Release your materials under a similar (more permissive) license (ShareAlike). That is, you should allow others to modify your version and publish their version.

You may not lift large passages of this work and claim them as your own, or publish them under a copyright-style license that forbids further modification. This work is meant to be part of a conversation about art and programming. Copyright stops the conversation, so I don't want it.

Acknowledgments

I won't attempt a complete list of thanks, but here's a good start:

- The entire community of SuperCollider users and developers, without whom, none of this. In particular:
 - James McCartney, for inventing the thing;

¹<http://www.levien.com/type/myfonts/inconsolata.html>

²<http://orgmode.org>

³<http://tug.org/texlive/>

⁴<https://bitbucket.org/rivanvx/beamer/wiki/Home>

⁵<http://creativecommons.org/licenses/by-sa/4.0/>

- Julian Rohrer and Alberto de Campo, for considering improvements to JITLib that make this modular-synthesis usage much easier.
- The *org-mode* development team, for a spectacularly customizable authoring tool (plus calendar, to-do list and in-general everything tool).
- Users of <http://tex.stackexchange.com> for advice on a few sticky \LaTeX problems.