



C o m m u n i t y   E x p e r i e n c e   D i s t i l l e d

# Mapping and Visualization with SuperCollider

Create interactive and responsive audio-visual applications with  
SuperCollider

**Marinos Koutsomichalis**

**[PACKT]** open source\*  
PUBLISHING community experience distilled

# Mapping and Visualization with SuperCollider

Create interactive and responsive audio-visual applications with SuperCollider

**Marinos Koutsomichalis**



BIRMINGHAM - MUMBAI

# Mapping and Visualization with SuperCollider

Copyright © 2013 Packt Publishing

All rights reserved. No part of this book may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, without the prior written permission of the publisher, except in the case of brief quotations embedded in critical articles or reviews.

Every effort has been made in the preparation of this book to ensure the accuracy of the information presented. However, the information contained in this book is sold without warranty, either express or implied. Neither the authors, nor Packt Publishing, and its dealers and distributors will be held liable for any damages caused or alleged to be caused directly or indirectly by this book.

Packt Publishing has endeavored to provide trademark information about all of the companies and products mentioned in this book by the appropriate use of capitals. However, Packt Publishing cannot guarantee the accuracy of this information.

First published: November 2013

Production Reference: 1191113

Published by Packt Publishing Ltd.  
Livery Place  
35 Livery Street  
Birmingham B3 2PB, UK.

ISBN 978-1-78328-967-7

[www.packtpub.com](http://www.packtpub.com)

Cover Image by Aniket Sawant ([aniket\\_sawant\\_photography@hotmail.com](mailto:aniket_sawant_photography@hotmail.com))

# Credits

**Author**

Marinos Koutsomichalis

**Project Coordinator**

Joel Goveya

**Reviewers**

João Martinho Moura

Joshua Parmenter

Phil Thomson

**Proofreaders**

Mario Cecere

Stephen Copestake

**Indexer**

Monica Ajmera Mehta

**Acquisition Editor**

Vinay Argekar

**Graphics**

Ronak Dhruv

Abhinash Sahu

**Commissioning Editor**

Poonam Jain

**Production Coordinator**

Pooja Chiplunkar

**Technical Editors**

Kunal Anil Gaikwad

Iram Malik

Shruti Rawool

**Cover Work**

Pooja Chiplunkar

**Copy Editors**

Roshni Banerjee

Gladson Monteiro

Deepa Nambiar

Karuna Narayanan

Shambhavi Pai

# About the Author

**Marinos Koutsomichalis** (Athens, 1981) is an artist and scholar working with sound and a wide range of other media. His artistic work interrogates the specifics of site, perception, technology, and material. His academic interests include computer programming, generative art, new aesthetics, and environmental sound and noise. He has widely performed, exhibited, and lectured internationally and has held residencies in miscellaneous research centers and institutions. He has an MA by research in composition with digital media by the University of York and, as of writing, he is a candidate PhD in Music, Sound, and Media Art at the De Montfort University. He is in the board of the Contemporary Music Research Center (KSYME-CMRC) and also the director of its class of Electronic Music and Sound Synthesis. As of writing, he is a research fellow in the University of Turin.

---

I would like to thank Packt Publishing for offering me the amazing opportunity to write this book, and in particular Shreerang Deshpande, Joel Goveya, Poonam Jain, Kunal Anil Gaikwad, Iram Malik, and Shruti Rawool for guiding me through the complexities of such a task. I would also like to thank the reviewers of this title, namely Josh Parmenter, João Martinho Moura, and Phil Thomson, for their invaluable comments and suggestions. I would also like to thank my parents, Anna and Georgios, as well as my sister, Danai, for their long term understanding and support. Part of this book was written in Milatos, North Crete, while being accommodated by my partner's parents, Maria and Michalis, who deserve a special mention for making me feel comfortable during my stay there. Last but not least, I would like to express my profound gratitude towards my partner, Phaedra Logariastaki, for her unconditional support and for spending the whole of her summer vacations watching me sitting in front of a laptop instead of being with her. Without the support of all these people, this book would have been impossible to finish.

---

# About the Reviewers

**João Martinho Moura** is a researcher and media artist born in Portugal. His interests lie in digital art, intelligent interfaces, digital music, and computational aesthetics. He was invited as a professor at the Master Program in Technology and Digital Arts at the University of Minho, Portugal, teaching Programming for Digital Arts.

In 2013, he received the *National Multimedia Award-Art & Culture* from the APMP Multimedia Association in Portugal.

He has presented his work and research in a variety of conferences related to arts and technology, including:

- The International Festival for the Post-Digital Creation Culture OFFF (2008)
- World Congress on Communication and Arts (2010)
- SHiFT – Social and Human Ideas for Technology (2009)
- International Symposium on Computational Aesthetics in Graphics, Visualization, and Imaging CAe (2008)
- ARTECH (2008)
- ARTECH (2010)
- Computer Interaction (2009)
- ZON Digital Games (2007)
- International Creative Arts Fair (2008)
- ZON Multimédia Premium (2008)
- Le Corps Numérique-entre Culturel Saint-Exupéry (2011)
- Semibreve Award (2012)
- TEI International Conference on Tangible, Embedded, and Embodied Interaction (2011)
- Guimarães European Capital of Culture 2012

- Bodycontrolled Series LEAP – Lab for Electronic Arts and Performance Berlin (2012)
- Future Places (2012)
- The Ars Electronica Animation Festival (2012)
- SLSA Conference-Society for Literature, Science, and the Arts (2013),  
xCoAx – Computation Communication Aesthetics and X (2013)

His work has been presented in a variety of places in Portugal, Italy, USA, Brazil, UK, France, Hong Kong, Belgium, Germany, Israel, Spain, and Austria.

He is a researcher at engageLab, a laboratory at the intersection of arts and technology, established by two research centers at University of Minho, the Centre for Communication and Society Studies and the Centre Algoritmi.

---

I would like to thank the engageLab laboratory, at University of Minho, with a special mention to Pedro Branco and Nelson Zagalo.

---

**Joshua Parmenter** is a composer and performer of contemporary music with a focus on interactive live electronics. His works have been performed throughout America and Europe. Over the past decade, he has also been one of the developers in the open-source SuperCollider project. He also contributed to the *SuperCollider Book* available from MIT Press.

**Phil Thomson** is a Vancouver-based listener, composer, and writer/editor. His works have been heard in concerts and broadcasts in Canada, US, and abroad. His works for dance routines have been integrated with performances by choreographers, such as Jennifer Clarke Arora, James Gnam, and Sara Coffin. His writings have been published online by the Canadian Electroacoustic Community and in print by the Cambridge University Press.

# www.PacktPub.com

## Support files, eBooks, discount offers and more

You might want to visit [www.PacktPub.com](http://www.PacktPub.com) for support files and downloads related to your book.

Did you know that Packt offers eBook versions of every book published, with PDF and ePub files available? You can upgrade to the eBook version at [www.PacktPub.com](http://www.PacktPub.com) and as a print book customer, you are entitled to a discount on the eBook copy. Get in touch with us at [service@packtpub.com](mailto:service@packtpub.com) for more details.

At [www.PacktPub.com](http://www.PacktPub.com), you can also read a collection of free technical articles, sign up for a range of free newsletters and receive exclusive discounts and offers on Packt books and eBooks.



<http://PacktLib.PacktPub.com>

Do you need instant solutions to your IT questions? PacktLib is Packt's online digital book library. Here, you can access, read and search across Packt's entire library of books.

## Why Subscribe?

- Fully searchable across every book published by Packt
- Copy and paste, print and bookmark content
- On demand and accessible via web browser

## Free Access for Packt account holders

If you have an account with Packt at [www.PacktPub.com](http://www.PacktPub.com), you can use this to access PacktLib today and view nine entirely free books. Simply use your login credentials for immediate access.





# Table of Contents

<b>Preface</b>	<b>1</b>
<b>Chapter 1: Scoping, Plotting, and Metering</b>	<b>7</b>
<b>Plotting audio, numerical datasets, and functions</b>	<b>7</b>
Using plot and plot graph	8
Using plotter	10
Using SoundFileView	12
<b>Scoping signals</b>	<b>13</b>
Scoping waveforms	14
Scoping spectra	16
<b>Metering levels</b>	<b>17</b>
Monitoring signals	17
Monitoring numerical data	19
<b>Nonstandard and complex visualizers</b>	<b>20</b>
Nonstandard visualizers	20
A complex scope	21
<b>Summary</b>	<b>22</b>
<b>Chapter 2: Waveform Synthesis</b>	<b>23</b>
<b>Waveform synthesis fundamentals</b>	<b>24</b>
<b>Time domain representation</b>	<b>24</b>
Waveform species	26
DC, amplitude, frequency, and phase	27
<b>Custom waveform generators</b>	<b>29</b>
Wavetable lookup synthesis	29
Using envelopes as wavetables	31
Custom aperiodic waveform generators	32
<b>Waveform transformations</b>	<b>33</b>
<b>Waveshaping</b>	<b>34</b>
Unary operations	36

Binary operations	38
Bitwise operations	39
<b>Summary</b>	<b>41</b>
<b>Chapter 3: Synthesizing Spectra</b>	<b>43</b>
<b>Introducing the frequency domain</b>	<b>43</b>
Spectra	44
Fast Fourier Transform in SuperCollider	45
<b>Creating and manipulating spectra</b>	<b>46</b>
Aggregating and enriching spectra	46
Sculpting and freezing spectra	48
Shifting, stretching, and scrambling spectra	50
Using the pvcalc method	52
<b>Visualizing spectra</b>	<b>53</b>
Limitations of spectral scoping	53
Optimizing spectra for scoping	54
<b>Summary</b>	<b>56</b>
<b>Chapter 4: Vector Graphics</b>	<b>57</b>
<b>Learning the vector graphics fundamentals</b>	<b>58</b>
Drawing primitive shapes and loading images	59
Complex shapes and graphics state	60
Introducing colors, transparency, and gradients	61
<b>Abstractions and models</b>	<b>63</b>
Objects and prototypes	64
Factories	65
Geometrical transformations, matrices, and trailing effects	68
<b>Complex structures</b>	<b>71</b>
Particle systems	71
Fractals	74
<b>Summary</b>	<b>79</b>
<b>Chapter 5: Animation</b>	<b>81</b>
<b>Fundamentals of motion</b>	<b>81</b>
Motion species	82
Using UserView	82
Animating complex shapes and sprites	84
<b>Fundamental animation techniques</b>	<b>85</b>
Trailing effects	85
Interaction and event-driven programming	86
Particle systems	88
<b>Advanced concepts</b>	<b>90</b>
Animating fractals	91

---

Adding dynamics to simulate physical forces	94
Kinematics	98
<b>Summary</b>	<b>101</b>
<b>Chapter 6: Data Acquisition and Mapping</b>	<b>103</b>
<b>Data acquisition</b>	<b>104</b>
Dealing with local files	104
Accessing data remotely	107
Using OSC	109
Using MIDI	112
Using Serial Port	113
<b>Machine listening</b>	<b>115</b>
Tracking amplitude and loudness	117
Tracking frequency	118
Timbre analysis and feature detection	119
Onset detection and rhythmical analysis	120
<b>Basic mappings</b>	<b>121</b>
Preparing and preprocessing data on the client side	122
Preparing and preprocessing data on the server side	124
Basic encodings and interpolation schemes	126
Sharing and distributing data	128
<b>Summary</b>	<b>130</b>
<b>Chapter 7: Advanced Visualizers</b>	<b>131</b>
<b>Audio visualizers</b>	<b>131</b>
Trailing waveforms	132
Spectrogram	133
<b>Music visualizers</b>	<b>136</b>
Rotating windmills	137
Kinematic patterns	138
<b>Visualizing and sonifying data</b>	<b>140</b>
Particles and grains	141
Fractalizer	144
<b>Summary</b>	<b>148</b>
<b>Chapter 8: Intelligent Encodings and Automata</b>	<b>149</b>
<b>Analyzing data</b>	<b>150</b>
Statistical analyses and metadata	150
Probabilities and histograms	152
Dealing with textual datasets	153
<b>Advanced mappings</b>	<b>156</b>
Complex and intelligent encodings	156
Neural networks	159

---

<b>Automata</b>	<b>162</b>
Cellular automata	163
Game of Life	166
<b>Summary</b>	<b>169</b>
<b>Chapter 9: Design Patterns and Methodologies</b>	<b>171</b>
<b>Blackboard</b>	<b>172</b>
Methodology	172
Model-View-Controller	174
Handling multiple files and environments	176
Threads, semaphores, and guards	179
<b>The View</b>	<b>182</b>
Clients and interfaces	182
Implementation	184
Strategies and policies	186
<b>The Model</b>	<b>188</b>
Aggregates and wrappers	188
Software agents	190
Introducing software actors and finalizing the model	192
<b>The Controller</b>	<b>193</b>
Game of Life	194
Finalizing the Controller	197
<b>Summary</b>	<b>199</b>
<b>Index</b>	<b>201</b>

---

# Preface

Welcome to the *Mapping and Visualization with SuperCollider* book. As of this writing, SuperCollider is almost two decades old and has already proven itself as a solid, state-of-the-art environment for all sorts of audio-oriented applications. Albeit, SuperCollider is primarily known as a sound synthesis environment; it does feature a powerful graphics engine and, to a certain extent, is an excellent choice for prototyping and implementing visual and audiovisual applications. This may come as a surprise to some, given that there does exist an abundance of specialized environments and frameworks out there; many of them are also more featured and optimized than SuperCollider will ever be. Nonetheless, and at least as far as visualization is concerned, the latter constitutes a very rational choice, as it exhibits several advantages over the former. Namely, it features one of the most powerful sound synthesis engines available on the planet; it has a powerful interpreted, dynamic, object-oriented, and quite easy-to-learn high-level programming language; it has built-in features to facilitate algorithmic music composition, which can easily integrate with computer graphics; it is easy to learn and use compared to other specialized frameworks; and it is relatively fast and stable.

This book pinpoints mapping and visualization with SuperCollider. It elaborates both fundamental and more advanced techniques and illustrates how SuperCollider can offer solutions to a wide range of typical mapping/visualization scenarios, varying from very rudimentary to highly complex ones. The explicit focus herein is mapping and visualization, yet a wide range of prerequisites, or merely relevant to the latter topics are discussed; these include sonification, generative art, statistical analysis, communication protocols, automata, and neural networks. These are all approached practically and from a hands-on perspective through numerous examples. Notwithstanding, theoretical issues are also discussed whenever appropriate, so that the reader develops a more in-depth understanding of the various topics. Throughout this book, the importance of object modeling is explicitly highlighted too, and software architecture itself is elaborated upon. In general, these are very important aspects of programming and given the minimal, or even nonexistent, presence of relevant resources regarding SuperCollider. This book aspires to be interesting to all seasoned and casual SuperCollider users.

## What this book covers

*Chapter 1, Scoping, Plotting, and Metering*, examines basic built-in scoping, plotting, and the metering of waveforms, signals, and numerical datasets in SuperCollider. In this chapter, we will discuss how to visualize numerical datasets, signals, and functions; how to scope waveforms and spectra in real time; how to monitor audio levels and numerical data; and how to implement more complex and nonstandard visualizers using various built-in GUI elements.

*Chapter 2, Waveform Synthesis*, elaborates various waveform synthesis techniques, with emphasis on the visual, rather than acoustic, aspects of audio. In this chapter, we will discuss waveform synthesis fundamentals and learn how to generate custom and good-looking (in any subjective way) waveforms based on a series of techniques.

*Chapter 3, Synthesizing Spectra*, is similar in spirit to the *Chapter 2, Waveform Synthesis*, yet it deals with spectra rather than with waveforms. In this chapter, we will focus on the visual aspects of audio spectra and learn to synthesize custom and good-looking (again in any subjective way) spectra using a variety of both time-domain and frequency-domain techniques.

*Chapter 4, Vector Graphics*, deals with vector graphics and discusses both fundamental theoretical concepts as well as how to create static drawings of arbitrary complexity in SuperCollider using a wide range of techniques. Color, matrix operations, as well as complex visual structures such as particle systems and fractals are discussed. In this chapter, we will also discuss object modeling with `Event` and the factory design pattern.

*Chapter 5, Animation*, elaborates on video animation. Therein, we will demonstrate how to implement different kinds of motion, how to create trailing effects, as well as how to animate complex visual structures and systems. We will also introduce ourselves with more advanced techniques, such as emulating environmental forces and real-life systems or designing articulated bodies using kinematics.

*Chapter 6, Data Acquisition and Mapping*, explains how arbitrary real-world numerical data can be retrieved, accessed, processed, and used in SuperCollider. This chapter elaborates on machine listening (that is, how to extract information out of audio signals) and discusses basic mappings and encodings.

*Chapter 7, Advanced Visualizers*, elaborates on a series of advanced examples wherein audio and data are visualized/sonified in various ways. The examples range from trailing waveforms and spectrogram implementations to more imaginative ones featuring kinematic structures, fractals, and particle systems.

*Chapter 8, Intelligent Encodings and Automata*, serves as an introduction to more advanced topics such as statistical analysis, textual parsing, advanced encodings, neural networks, and cellular automata. Therein we will also discuss a possible implementation of the famous game of life automaton.

*Chapter 9, Design Patterns and Methodologies*, discusses software architecture and explains how certain design patterns and methodologies can be used by programmers and computer scientists to solve certain recurring problems. Starting with the requirements for a quite complex generative project, we will proceed step-by-step, designing and materializing it in an efficient and conceptually understandable way.

## What you need for this book

To use the code provided with this book, you need the latest version of the SuperCollider programming environment, which may be downloaded from <http://supercollider.sourceforge.net/downloads/>. As of this writing, Version 3.6.5 is the official stable release, while 3.7 is still under development. While some of the code heretofore is backward compatible with older versions of the program, the reader is encouraged to use version 3.6.5 or newer. Bear in mind that the GUI part of SuperCollider, on which this book relies a lot, has been substantially changed over the last major updates and, thus, you need to have at least Version 3.5 installed. Other than the SuperCollider programming environment, several examples rely on the SC3-plugins library, which can be downloaded from <http://sourceforge.net/projects/sc3-plugins/>. In some exceptional cases, you will also have to install some Quark extensions or even third-party softwares; these are all discussed in the relevant chapters. Yet, you should know how to use the Quarks system. Finally, for those examples that depend on a working Internet connection, you should make sure that your computer has access to it.

To make the best of this book, it is both expected and assumed that you are already familiar with the fundamentals of sound synthesis and have some experience with SuperCollider. In particular, you should be comfortable with variables, `SynthDef`/`Synths`, functions, routines, object-oriented programming, writing classes, scheduling, client/server architecture, and so on. Those readers who are not sure whether their SuperCollider skills are sufficient are encouraged to study the tutorials found on <http://supercollider.sourceforge.net/learning/>.



## Who this book is for

This book is for intermediate and advanced SuperCollider users who are interested in mapping and visualization for either scientific or artistic applications. Care has been taken to ensure that the book will be of interest to artists as well as scientists and other specialists, and even to those only indirectly interested in mapping/visualization. The book also discusses a wide range of topics related, but not specific, to the latter, such as automata, generative art, animation, artificial neural networks, and others, and may, therefore, be of interest to anyone having interest in those fields. To some extent, this book is also of interest to all SuperCollider users, including seasoned and new ones because it addresses object modeling and software-architecture-specific topics. Therefore, it provides the necessary background to all those interested in materializing more complex projects of any nature. The primary audience of this book is expected to be artists, scientists, and other specialists interested in mapping, visualization, and generative audiovisual systems.

## Conventions


In this book, you will find a number of styles of text that distinguish between different kinds of information. Here are some examples of these styles, and an explanation of their meaning.


Code words in text are shown as follows: "Whenever they are invoked, a parent `Window` is created containing an instance of `Plotter` whose specifics are configured accordingly."

A block of code is set as follows:

```
( // MyfancyStereoScope Example
Server.default.waitForBoot({ // wait for server to boot
  MyFancyStereoScope.new();
  {[Saw.ar(400), Saw.ar(402)]}.play(a)
})
)
```

**New terms** and **important words** are shown in bold. Words that you see on the screen, in menus or dialog boxes for example, appear in the text like this: "In order to make this code work, we also need to load the `StandardFirmata` code in our Arduino, which we can find in the **Examples | Firmata** submenu of the Arduino **Integrating Development Environment (IDE)**".

[  Warnings or important notes appear in a box like this. ]

[  Tips and tricks appear like this. ]

## Reader feedback

Feedback from our readers is always welcome. Let us know what you think about this book – what you liked or may have disliked. Reader feedback is important for us to develop titles that you really get the most out of.

To send us general feedback, simply send an e-mail to [feedback@packtpub.com](mailto:feedback@packtpub.com), and mention the book title via the subject of your message.

If there is a topic that you have expertise in and you are interested in either writing or contributing to a book, see our author guide on [www.packtpub.com/authors](http://www.packtpub.com/authors).

## Customer support

Now that you are the proud owner of a Packt book, we have a number of things to help you to get the most from your purchase.

## Downloading the example code

You can download the example code files for all Packt books you have purchased from your account at <http://www.packtpub.com>. If you purchased this book elsewhere, you can visit <http://www.packtpub.com/support> and register to have the files e-mailed directly to you.

## Errata

Although we have taken every care to ensure the accuracy of our content, mistakes do happen. If you find a mistake in one of our books – maybe a mistake in the text or the code – we would be grateful if you would report this to us. By doing so, you can save other readers from frustration and help us improve subsequent versions of this book. If you find any errata, please report them by visiting <http://www.packtpub.com/submit-errata>, selecting your book, clicking on the **erratasubmissionform** link, and entering the details of your errata. Once your errata are verified, your submission will be accepted and the errata will be uploaded on our website, or added to any list of existing errata, under the Errata section of that title. Any existing errata can be viewed by selecting your title from <http://www.packtpub.com/support>.

## Piracy

Piracy of copyright material on the Internet is an ongoing problem across all media. At Packt, we take the protection of our copyright and licenses very seriously. If you come across any illegal copies of our works, in any form, on the Internet, please provide us with the location address or website name immediately so that we can pursue a remedy.

Please contact us at [copyright@packtpub.com](mailto:copyright@packtpub.com) with a link to the suspected pirated material.

We appreciate your help in protecting our authors, and our ability to bring you valuable content.

## Questions

You can contact us at [questions@packtpub.com](mailto:questions@packtpub.com) if you are having a problem with any aspect of the book, and we will do our best to address it.

# 1

## Scoping, Plotting, and Metering

Visualizing audio signals and numerical datasets can be very straightforward in SuperCollider with the built-in scoping, plotting, and metering functionalities. The corresponding GUI objects are simple to use, yet they are highly customizable and extremely powerful. In this chapter we will introduce a series of fundamental techniques, and learn how to design both basic as well as more advanced custom visualizers. However, it should be noted that all the examples herein assume normalized datasets and test signals, deferring the complexities of data mapping and signal optimization to be discussed in depth in subsequent chapters.

The topics that will be covered in this chapter are as follows:

- Plotting audio, numerical datasets, and functions
- Scoping waveforms and spectra
- Metering signals and data
- Nonstandard and complex visualizers

### **Plotting audio, numerical datasets, and functions**

Before discussing how we can scope audio signals in real time, it is worth reviewing the various ways in which we can create static graphs and charts out of arbitrary numerical datasets or signals.

## Using plot and plot graph

SuperCollider provides us with a very handy `plot` method. We can use this method in different situations to create graphs on the fly from instances of `Function`, `ArrayedCollection`, `Env`, `Buffer`, `SoundFile`, `WaveTable`, and from a series of other objects (also depending on what extensions we have installed). An example of this is shown in the following code:

```
{SinOsc.ar(100)}.plot(0.1);           // plot a 0.1 seconds of a
sinewave
[5,10,100, 50, 60].plot;              // plot a numerical dataset
Env([0,1,0],[1,1],[-10,2]).plot;      // plot an envelope
Signal[0,1,0.5,1,0].plot;            // plot a signal
Wavetable.chebyFill(513,[1]).plot;    // plot a wavetable

( // plot the contents of a sound file
Server.default.waitForBoot({ // wait for Server to boot
  Buffer.read(Server.default, Platform.resourceDir +/+
    "sounds/all1wk01.wav").plot;
});
)
```

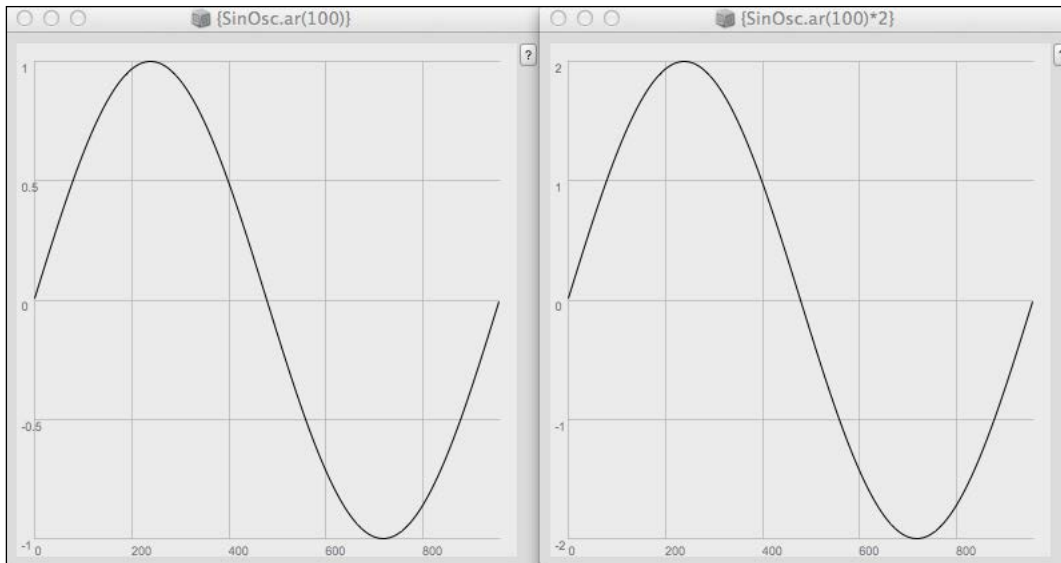
### Downloading the example code



You can download the example code files for all Packt books you have purchased from your account at <http://www.packtpub.com>. If you purchased this book elsewhere, you can visit <http://www.packtpub.com/support> and register to have the files e-mailed directly to you.

In all cases, the resulting graphs will be automatically normalized with respect to the kind of data plotted so that each dimensions' display range is determined by the minimum and maximum quantities it has to represent; that is, to say that the plot's graph is *content-dependent*. Additionally, their meaning depends upon the receiver (that is, the kind of object plotted) so that for instances of `Array`, `Wavetable`, or `Signal`, the graph would represent the value per index; for `UGen` graphs, amplitude per unit time; for instances of `Env`, value per unit time; and for instances of `Buffer`, amplitude per frame. Since its behavior is different for different kinds of objects, the plot is said to be **polymorphic**. We should always consider the implicit consequences of these two properties. For example, the following two waveforms could be easily mistaken as identical, even if they are not:

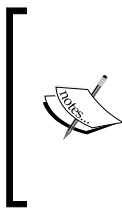
```
( // plot two sinusoids of different amplitude
{SinOsc.ar(100)}.plot(bounds:Rect(0,0,400,400));
{SinOsc.ar(100)*2}.plot(bounds:Rect(400,0,400,400));
)
```



To compensate for such a phenomenon, we need to explicitly set the minima (*minval*) and maxima (*maxval*) arguments. Interestingly enough, we can also plot abstract functions as long as they are one-argument ones and return some arithmetic value. We can do this with the `plotGraph` method, as follows:

```
{arg x; tan(x**2);}.plotGraph(100,-pi,pi); // graph out of a function
```

Here, the interpreter calculates the output of the given function for 100 different values in the range of  $\pm \pi$  and populates the graph with the results; the horizontal axis representing node indexes and the vertical axis representing the function's output.



Buffer objects have a finite capacitance measured in frames; each frame may hold exactly one sample, therefore, a frame is the container of a sample.

**Polymorphism** in Computer Science refers to the ability in programming to present the same interface for different underlying forms.

## Using plotter

Both `plot` and `plotGraph` are convenient methods, which ostensibly are just abstractions of a series of tasks. Whenever they are invoked, a parent window is created containing an instance of `Plotter` whose specifications are configured accordingly. Explicitly creating and using `Plotter` allows sophisticated control over the way our data is plotted. The following code exemplifies a number of features of the `Plotter` object:

```
( // data visualization using custom plotters
// the parent window
var window = Window.new("Plotter Example", Rect(0,0,640,480)).front;

// the datasets to visualize
var datasetA = Array.fill(1000,{rrand(-1.0,1.0)}); // random floats
var datasetB = [ // a 2-dimensional array of random floats
  Array.fill(10,{rrand(-1.0,1.0)}),
  Array.fill(10,{rrand(-1.0,1.0)})
];

// the plotters
var plotterA = Plotter("PlotterA",Rect(5,5,630,235),window);
var plotterB = Plotter("PlotterB",Rect(5,240,630,235),window);

// setup and customize plotterA
plotterA.value_(datasetA); // load dataset
plotterA.setProperties( // customize appearance
  \plotColor, Color.red, // plot color
  \backgroundColor, Color.black, // background color
  \gridColorX, Color.white, // gridX color
  \gridColorY, Color.yellow) // gridY color
.editMode_(true) // allow editing with the cursor
.editFunc_({ // this function is evaluated whenever data is edited
  arg plotter,plotIndex,index,val,x,y;
  ("Value: " ++ val ++ " inserted at index: " ++ index ++
   ".").println;
});

// setup and customize plotterB
plotterB.value_(datasetB); // load datasetB
plotterB.superpose_(true); // allow channels overlay
plotterB.setProperties(
```