



# **D3** Data-Driven Documents

## **POCKET PRIMER**



*CD-ROM Included!*



OSWALD CAMPESATO

# D3

*Pocket Primer*

---

## **LICENSE, DISCLAIMER OF LIABILITY, AND LIMITED WARRANTY**

By purchasing or using this book and disc (the “Work”), you agree that this license grants permission to use the contents contained herein, including the disc, but does not give you the right of ownership to any of the textual content in the book / disc or ownership to any of the information or products contained in it. *This license does not permit uploading of the Work onto the Internet or on a network (of any kind) without the written consent of the Publisher.* Duplication or dissemination of any text, code, simulations, images, etc. contained herein is limited to and subject to licensing terms for the respective products, and permission must be obtained from the Publisher or the owner of the content, etc., in order to reproduce or network any portion of the textual material (in any media) that is contained in the Work.

MERCURY LEARNING AND INFORMATION (“MLI” or “the Publisher”) and anyone involved in the creation, writing, or production of the companion disc, accompanying algorithms, code, or computer programs (“the software”), and any accompanying Web site or software of the Work, cannot and do not warrant the performance or results that might be obtained by using the contents of the Work. The author, developers, and the Publisher have used their best efforts to insure the accuracy and functionality of the textual material and/or programs contained in this package; we, however, make no warranty of any kind, express or implied, regarding the performance of these contents or programs. The Work is sold “as is” without warranty (except for defective materials used in manufacturing the book or due to faulty workmanship).

The author, developers, and the publisher of any accompanying content, and anyone involved in the composition, production, and manufacturing of this work will not be liable for damages of any kind arising out of the use of (or the inability to use) the algorithms, source code, computer programs, or textual material contained in this publication. This includes, but is not limited to, loss of revenue or profit, or other incidental, physical, or consequential damages arising out of the use of this Work.

The sole remedy in the event of a claim of any kind is expressly limited to replacement of the book and/or disc, and only at the discretion of the Publisher. The use of “implied warranty” and certain “exclusions” vary from state to state, and might not apply to the purchaser of this product.

# D3

## *Pocket Primer*

---

Oswald Campesato



MERCURY LEARNING AND INFORMATION

Dulles, Virginia

Boston, Massachusetts

New Delhi

Copyright ©2016 by MERCURY LEARNING AND INFORMATION LLC. All rights reserved.

*This publication, portions of it, or any accompanying software may not be reproduced in any way, stored in a retrieval system of any type, or transmitted by any means, media, electronic display or mechanical display, including, but not limited to, photocopy, recording, Internet postings, or scanning, without prior permission in writing from the publisher.*

Publisher: David Pallai  
MERCURY LEARNING AND INFORMATION  
22841 Quicksilver Drive  
Dulles, VA 20166  
info@merclearning.com  
www.merclearning.com  
1-800-232-0223

O. Campesato. *D3 Pocket Primer*.  
ISBN: 978-1-938549-65-6

The publisher recognizes and respects all marks used by companies, manufacturers, and developers as a means to distinguish their products. All brand names and product names mentioned in this book are trademarks or service marks of their respective companies. Any omission or misuse (of any kind) of service marks or trademarks, etc. is not an attempt to infringe on the property of others.

Library of Congress Control Number: 2014937375

151617321 Printed in the United States of America  
This book is printed on acid-free paper.

Our titles are available for adoption, license, or bulk purchase by institutions, corporations, etc.

For additional information, please contact the Customer Service Dept. at (800) 232-0223 (toll free). Digital versions of our titles are available at: [www.authorcloudware.com](http://www.authorcloudware.com) and other e-vendors.

The sole obligation of MERCURY LEARNING AND INFORMATION to the purchaser is to replace the book and/or disc, based on defective materials or faulty workmanship, but not based on the operation or functionality of the product.

*I'd like to dedicate this book to my parents –  
may this bring joy and happiness into their lives.*



# CONTENTS

<i>Preface</i> .....	<i>xiii</i>
----------------------	-------------

<b>Chapter 1: Introduction to D3</b> .....	<b>1</b>
What is D3? .....	1
D3 on Mobile Devices .....	2
D3 Boilerplate .....	2
Method Chaining in D3.....	3
The D3 Methods <code>select()</code> and <code>selectAll()</code> .....	3
Specifying UTF-8 in HTML5 Web Pages with D3 .....	4
Creating New HTML Elements .....	4
The Most Common Idiom in D3 .....	5
Binding Data to Document-Object-Model Elements .....	6
Generating Text Strings .....	8
Creating Simple Two-Dimensional Shapes .....	9
Bezier Curves and Text .....	12
Two-Dimensional Transforms.....	14
A Digression: Scaling Arrays of Numbers to Different Ranges .....	14
Tweening in D3 .....	16
Formatting Numbers .....	17
Working with Gradients .....	17
Linear Gradients .....	17
Radial Gradients.....	18
Adding HTML <code>&lt;div&gt;</code> Elements with Gradient Effects .....	19
Working with Portable Network Graphics Files.....	20
D3 Application Programming Interface Reference.....	21
Additional Code Samples on the CD .....	21
Summary.....	22



**Chapter 2: Arrays, Mouse Events, and Animation Effects ....25**

Finding the Maximum and Minimum Values in an Array.....	26
Working with Multidimensional Arrays .....	26
Two-Dimensional Arrays and Scatter Charts .....	29
D3 Data Scaling Functions.....	30
Other D3.js Scaling Functions .....	33
D3 Path Data Generator.....	33
What About this, \$this, and \$(this)? .....	35
D3 and Mouse Events .....	36
Mouse Events and Randomly Located Two-Dimensional Shapes .....	37
A “Follow the Mouse” Example .....	38
A Drag-and-Drop Example .....	39
Animation Effects with D3.....	41
Easing Functions in D3 .....	43
Zoom, Pan, and Rescale Effects with D3.....	44
Handling Keyboard Events with D3.....	44
Additional Code Samples on the CD .....	45
Summary.....	45

**Chapter 3: Working with Bar Charts in D3 .....47**

A Simple Horizontal Bar Chart .....	47
Rendering Horizontal and Vertical Axes with Labels.....	49
A Scaled Vertical Bar Chart with Labeled Axes.....	52
Using Date and Time Stamps to Label Axes.....	54
D3 Bar Charts with Unicode Characters .....	55
Bar Charts with Three-Dimensional Effects .....	57
Bar Charts with Filter Effects and Tooltips .....	61
Additional Filter Effects .....	63
Updating Bar Charts .....	67
Dynamically Adding and Removing Data From Bar Charts.....	71
Scrolling Animation Effects with Bar Charts.....	72
Additional Code Samples on the CD .....	73
Summary.....	75

**Chapter 4: Other Chart Types and Data Formats .....77**

Rendering A Line Graph .....	77
Rendering Multiple Nonlinear Graphs .....	80
Scatter Charts with Axes and Mouse Events .....	83
Selecting Equal Data Points in Scatter Charts .....	87
Rendering Pie Charts.....	88
A Histogram with Animation Effects .....	89
Working with Other Data Formats and Data Files .....	92
The XMLHttpRequest Request Object.....	92
The jQuery .ajax() Method .....	93
Useful D3 Methods for CSV Files .....	93
CSV: Synchronous Versus Nonsynchronous D3 Methods .....	94

Line Graphs with CSV Data and Mouse Events .....	94
Bar Charts with Three-Dimensional Effects from Comma-Separated-Value Files.....	99
Additional Code Samples on the CD .....	101
Summary.....	101

## **Chapter 5: SVG Essentials ..... 103**

Overview of SVG.....	104
Basic Two-Dimensional Shapes in SVG.....	104
SVG Gradients and the <path> Element.....	107
SVG <polygon> Element .....	109
Bezier Curves and Transforms .....	110
SVG Filters and Shadow Effects .....	114
Rendering Text Along an SVG <path> Element .....	115
SVG Transforms .....	116
The SVG <clipPath> Element .....	119
Other SVG Features .....	120
SVG Animation .....	121
Creating Three-Dimensional Effects in SVG .....	121
SVG and HTML.....	122
SVG and JavaScript.....	122
CSS3 and SVG.....	123
CSS3 and SVG Bar Charts.....	124
Similarities and Differences Between SVG And CSS3 .....	124
SVG and XSLT (Extensible Stylesheet Language Transformations) .....	125
Additional Code Samples on the CD .....	125
Summary.....	126

## **Chapter 6: Introduction to CSS3 Graphics and Animation ..... 127**

CSS3 Support and Browser-Specific Prefixes for CSS3 .....	128
Quick Overview of CSS3 Features.....	129
CSS3 Pseudoclasses, Attribute Selection, and Relational Symbols .....	129
CSS3 Pseudoclasses .....	130
CSS3 Attribute Selection .....	130
CSS3 Shadow Effects and Rounded Corners .....	131
Specifying Colors with Red/Green/Blue Triples and Hue/Saturation/Lightness Representations .....	131
CSS3 and Text Shadow Effects .....	132
CSS3 and Box Shadow Effects .....	135
CSS3 and Rounded Corners.....	135
CSS3 Gradients .....	137
Linear Gradients .....	137
Radial Gradients.....	140
CSS3 Two-Dimensional Transforms .....	142
Rotate Transforms.....	145

CSS3 Three-Dimensional Animation Effects.....	148
CSS3 Media Queries.....	151
CSS3 and SVG.....	152
Additional Code Samples on the CD .....	153
Summary.....	153

## **Chapter 7: D3 with CSS3, SVG, and HTML5 Canvas.....155**

D3 Code Samples with HTML5 Canvas.....	155
Updated CSS3 Stylesheets for this Chapter .....	156
D3 and CSS3 Effects .....	156
D3 and CSS3 Animation Effects .....	158
D3 and HTML5 Canvas .....	160
D3 and SVG .....	164
Bubble Charts with JSON Data .....	168
Additional Code Samples on the CD .....	170
Summary.....	171

## **Chapter 8: D3 with Ajax, HTML5 WebSockets, and NodeJS .....173**

D3 and Ajax Requests.....	173
D3 with PHP Data.....	177
D3 with MySQL Data.....	178
D3 Bar Charts with a WebSocket Server.....	180
D3 and NodeJS (Optional) .....	187
Inserting an <svg> Element in an HTML Web Page .....	187
Rendering SVG Graphics with D3 and NodeJS (Optional) .....	189
Additional Code Samples on the CD .....	191
Summary.....	191

## **Chapter 9: Miscellaneous D3 Application Programming Interfaces and Other Toolkits .....193**

Maps in D3 (Choropleth) .....	193
Adding Tooltips to a United States Map .....	196
D3 and Google Maps.....	198
GeoJSON and D3 TopoJSON .....	198
Other Maps.....	199
The D3 Force Layout .....	200
Using D3 Force with CSS Instead of SVG .....	203
D3 Trees .....	203
Voronoi Diagrams.....	205
Toolkits That are D3 Extensions .....	209
The ChartBuilder Extension .....	209
The CrossFilter Extension.....	209
The dc.js Extension.....	209
Rickshaw.....	210
D3 and Other Toolkits .....	211

D3 Plugins .....	211
DVL(Dynamic Visualization LEGO) for Data Visualization .....	212
Vega: A Visualization Grammar .....	212
NVD3.....	213
DexChart: Reusable Charts .....	213
R Programming with D3-Based Toolkits .....	213
Additional D3 APIs .....	213
The D3 Brushes API .....	214
D3 and HTML5 Web Audio.....	214
What About D3 for Three-Dimensional Graphics and Animation? .....	214
Other D3 Resources .....	215
Additional Code Samples on the CD .....	216
Summary.....	216

## **Chapter 10: HTML5 Mobile Applications on Android and iOS .....217**

HTML5/CSS3 and Android Applications .....	218
SVG and Android Applications.....	221
HTML5 Canvas and Android Applications.....	223
Android and HTML5 Canvas Multiline Graphs.....	225
What is PhoneGap?.....	228
How Does PhoneGap Work?.....	229
Software Dependencies for PhoneGap 3.0.....	229
Creating Android Hybrid Applications with PhoneGap 3.0.....	230
Creating iOS Hybrid Applications with PhoneGap 3.0.....	233
Requirements for Deploying Mobile Apps to iOS Devices .....	233
Rendering a CSS3 Cube on iOS Using PhoneGap.....	234
D3 and Android Applications .....	237
D3 and iOS Applications .....	239
Developing D3-Based Mobile Applications for Google Glass .....	241
How does Google Glass Work? .....	241
Supported HTML5 Tags.....	242
Unsupported HTML5 Tags.....	242
Deploying Android Applications to Google Glass .....	242
Displaying Google Glass in an Emulator .....	243
Other Useful Links for Google Glass .....	244
Other Google Glass Code Samples .....	244
Additional Code Samples on the CD .....	244
Summary.....	244

## **Index .....245**

### **On The CD**

#### **Appendix A: Overview of SVG**

#### **Appendix B: Introduction to Android**

#### **Appendix C: HTML5 and JavaScript Toolkits**

#### **Appendix D: Rendering 2D Shapes in HTML5 Canvas**



# PREFACE

## WHAT IS THE PRIMARY VALUE PROPOSITION FOR THIS BOOK?

---

**T**his book endeavors to provide you with as much up-to-date information as possible that can be reasonably included in a book of this size. There are many unique features of this book, including two chapters that are dedicated to CSS3 (with 2D/3D graphics and animation and how to leverage SVG in CSS selectors) and SVG (with custom code samples), and also an Appendix for HTML5 Canvas. There are two additional chapters that are unique to this D3 book: one chapter is devoted to hybrid mobile applications with D3 (deployed to Android tablets Nexus 7 2 and Asus Prime and an iPad3) with version of PhoneGap 3.0 (currently the latest release), and another chapter contains integrated code samples with CSS3, HTML5 Canvas, and SVG.

Other unique features of this book include code samples that show you how to render 3D shapes (including 3D bar charts with animation), create multiple animation effects, and how to create SVG filters for your HTML Web pages. Moreover, you will see code samples that show you how to work with Unicode, CSV files, Ajax, HTML5 WebSockets, and an example involving NodeJS. Finally, the author has written multiple open source projects (links are provided toward the end of this Preface) containing literally thousands of code samples so that you can explore the capabilities of CSS3, SVG, HTML5 Canvas, and jQuery combined with CSS3.

## THE TARGET AUDIENCE

---

This book is intended to reach an international audience of readers with highly diverse backgrounds in various age groups. While many readers know how to read English, their native language is not always English (which could be their second, third, or even fourth language). Consequently, this book uses

standard English rather than colloquial expressions that might be confusing to those readers. As you know, many people learn by different types of imitation, which includes reading, writing, or hearing new material (yes, some basic videos are also available). This book takes these points into consideration in order to provide a comfortable and meaningful learning experience for the intended readers.

## GETTING THE MOST FROM THIS BOOK

---

Some programmers learn well from prose, others learn well from sample code (and lots of it), which means that there's no single style that can be used for everyone.

Moreover, some programmers want to run the code first, see what it does, and then return to the code to delve into the details (and others use the opposite approach).

Consequently, there are various types of code samples in this book: some are short, some are long, and other code samples “build” from earlier code samples.

The goal is to show (and not just tell) you a variety of visual effects that are possible, some of which you might not find anywhere else. You benefit from this approach because you can pick and choose the visual effects and the code that creates those visual effects.

## WHAT TYPE OF MOBILE APPLICATIONS ARE IN THIS BOOK?

---

The screenshots throughout this book are based on hybrid HTML5 mobile applications that were deployed to two Android tablets with Android 4.x and an iPad3, as well as some screenshots from a Chrome browser on a Macbook Pro. The final chapter in this book shows you how to use PhoneGap 3.0 to create D3-based hybrid HTML5 mobile applications for Android and iOS mobile devices. This book does not focus on “thin apps,” which refers to HTML Web pages that are accessed from a mobile device. However, you have the option of making all the code samples in this book available on a Website in order to access them on mobile devices.

## HOW WAS THE CODE FOR THIS BOOK TESTED?

---

The code samples in this book have been tested in a Google Chrome browser (version 28.0.1500.95) on a Macbook Pro with OS X 10.8.3. Unless otherwise noted, no special browser-specific features were used, which means that the code samples ought to work in Chrome on other platforms, and also in other modern browsers. Exceptions are due to limitations in the cross-platform availability of specific features of D3 itself. Although the code also works in several earlier versions of Chrome on a Macbook Pro, you need to

test the code on your platform and browser (especially if you are using Internet Explorer).

Another point to keep in mind is that all references to “Web Inspector” refer to the Web Inspector in Chrome, which differs from the Web Inspector in Safari. If you are using a different (but still modern) browser or an early version of Chrome, you might need to check online for the sequence of keystrokes that you need to follow to launch and view the Web Inspector. Navigate to this link for additional useful information:

<http://benalman.com/projects/javascript-debug-console-log/>

## **WHAT ABOUT GOOGLE’S FORK OF WEBKIT?**

---

As this book goes to print, Google made Blink available, which is its fork of the WebKit engine. The code base for Blink (which some people call “WebKit 2.0”) is already available in the version of Chrome that was used to test the code samples in this book. Although the code (and therefore the features) for Blink will eventually diverge from the initial code base, the D3 code samples will render the same in other WebKit-based browsers as they do in Chrome-based browsers.

## **WHAT DO I NEED TO KNOW FOR THIS BOOK?**

---

The most important prerequisite is familiarity with HTML Web pages and JavaScript. If you want to be sure that you can grasp the material in this book, glance through some of the code samples to get an idea of how much is familiar to you and how much is new for you.

## **WHAT ABOUT SVG?**

---

No prior knowledge of SVG is required in order to learn D3. Keep in mind that D3 and SVG use the same attribute names for the 2D shapes that they both support, so if you already know the SVG-based attributes, that knowledge is obviously helpful. Another point to keep in mind is that even though D3 is a JavaScript-based “layer” over SVG, you can also use D3 with HTML5 Canvas, which is discussed in detail in an appendix.

## **WHY DOES THIS BOOK HAVE 250 PAGES INSTEAD OF 500 PAGES?**

---

This book is part of a Pocket Primer series whose books are between 200 and 250 pages. Second, the target audience consists of readers ranging from beginners to intermediate in terms of their knowledge of HTML and JavaScript. During the preparation of this book, every effort has been made to accommodate those readers so that they will be adequately prepared to explore more advanced features of D3 during their self study.



## WHY SO MANY CODE SAMPLES IN THE CHAPTERS?

---

One of the primary rules of exposition of virtually any kind is “show, don’t tell.” While this rule is not taken literally in this book, it’s the motivation for showing first and telling second. You can decide for yourself if show-first-then-tell is valid in this book by performing a simple experiment: when you see the code samples and the accompanying graphics effects in this book, determine if it’s more effective to explain (“tell”) the visual effects or to show them. If the adage “a picture is worth a thousand words” is true, then this book endeavors to provide both the pictures and the words.

## WHY SO MANY CODE SAMPLES ON THE CD?

---



The CD contains more than 1,000 code samples that vary from simple to moderate complexity using different combinations of technologies. Specifically, there are D3-based code samples along with “pure” SVG and “pure” CSS3 samples. In addition, you’ll find code samples that combine D3 with SVG, D3 with CSS3, SVG and CSS3. The code samples that combine CSS3 and SVG create 3D animation effects, whereas the pure SVG samples only create static effects.

In the code samples that are similar (admittedly there are many on the CD), they are included as a convenience, and their purpose is to show you stylistic nuances. Obviously the value of those subtle differences is highly subjective and impossible to quantify, which means that each person will evaluate the code samples in a different manner. As a parenthetical yet relevant aside: in the art world there can be two very similar original paintings by two different artists, and yet one painting is valuable whereas the other has much lower perceived value. Putting aside the intrinsic value of the code samples on the CD, it’s likely that some of them appeal to you and not to other people (and vice versa).

Think of the code samples as “concept code” in the sense that they provide you with ideas that you can borrow and enhance even further with your own variations. This approach of progressively adding details to provide you with a “swatch-like” variety of code samples will accommodate a range of reading styles, and you can jump in wherever you feel most comfortable. Since there will be concepts that cannot be fully discussed (due to the limited page count), the inclusion of similar code samples will make it simpler for you to grasp the additional techniques during your independent study.

*The key points to keep in mind is that supplemental code samples on the CD show you stylistic variations and they save you the time required to create those samples, but you can choose to view them or to ignore them.*

## DOESN’T THE CD OBVIATE THE NEED FOR THIS BOOK?

---

The CD contains all the code samples to save you time and effort from the error-prone process of manually typing code into an HTML Web page. In addition, there are situations in which you might not have easy access to CD.

Furthermore, the code samples in the book provide explanations that are not available on the CD.

Finally, as mentioned earlier in this Preface, there are some introductory videos available that cover HTML5, CSS3, HTML5 Canvas, and SVG. Navigate to the publisher's Website to obtain more information regarding their availability.

## **DOES THIS BOOK CONTAIN PRODUCTION-LEVEL CODE SAMPLES?**

---

The primary purpose of the code samples in this book is to illustrate various features of the D3 toolkit. Clarity has higher priority than writing more compact code that is more difficult to understand (and possibly more prone to bugs). If you decide to use any of the code in this book in a production Website, you ought to subject that code to the same rigorous analysis as the other parts of your HTML Web pages.

## **WHICH VERSION OF D3 IS FOR THIS BOOK?**

---

The code samples in this book use version 3.0 of D3 (released in December, 2012), which is currently the latest version of D3. If you plan to use an older version of D3, you need to test the D3 code samples in this book to ensure that they work with that older version.

## **TERMINOLOGY IN THIS BOOK**

---

You will see variations in the way that D3 methods are described. For example, you will see “the D3 method `.transition()`” and “the method `d3.transition()`” used interchangeably in this book. As another example, you will see “an HTML `<script>` element” and “a `<script>` element,” both of which refer to the same thing.

## **OTHER RELATED BOOKS BY THE AUTHOR**

---

1. HTML5 Canvas and CSS3:  
<http://www.amazon.com/HTML5-Canvas-CSS3-Graphics-Primer/dp/1936420341>
2. jQuery, HTML5, and CSS3:  
<http://www.amazon.com/jquery-HTML5-Mobile-Desktop-Devices/dp/1938549031>
3. HTML5 Pocket Primer:  
<http://www.amazon.com/HTML5-Pocket-Primer-Oswald-Campesato/dp/1938549104>
4. jQuery Pocket Primer:  
<http://www.amazon.com/dp/1938549147>

Upcoming book by the author:

[http://www.amazon.com/HTML5-Canvas-Pocket-Oswald-Campesato/dp/1938549678/ref=sr\\_1\\_2?ie=UTF8&qid=1434753297&sr=8-2&keywords=html5+canvas+pocket+primer](http://www.amazon.com/HTML5-Canvas-Pocket-Oswald-Campesato/dp/1938549678/ref=sr_1_2?ie=UTF8&qid=1434753297&sr=8-2&keywords=html5+canvas+pocket+primer)

The following open source projects contain code samples that supplement the material in various chapters of this book:

<https://github.com/ocampesato/angular-graphics>

<https://github.com/ocampesato/css3-graphics>

<https://github.com/ocampesato/d3-graphics>

<https://github.com/ocampesato/html5-graphics>

<https://github.com/ocampesato/jquery-css3-graphics>

<https://github.com/ocampesato/raphael-graphics>

<https://github.com/ocampesato/reactjs-graphics>

<https://github.com/ocampesato/svg-filters-graphics>

<https://github.com/ocampesato/svg-graphics>

# INTRODUCTION TO D3

This chapter introduces you to D3 and provides a collection of short code samples that illustrate how to use some useful D3 application programming interfaces (API). This chapter moves quickly, so even if you are already familiar with D3 it's worth your while to read (or at least skim through) the material in this chapter. The code samples in subsequent chapters use many of the APIs that are discussed in this chapter.

The first part of this chapter provides a brief description of the D3 toolkit and a list of some companies that use D3. The second part of this chapter shows you how to use some basic D3 methods by rendering simple two-dimensional (2D) shapes. In addition, you will learn how to create linear gradients and radial gradients.

---

**NOTE** *Be sure to launch the HTML Web pages in a browser as you read code samples in this book because this will show you what the code actually does, and it will also save you time understanding the code.*

---

## WHAT IS D3?

Mike Bostock created the open-source toolkit `Protovis`, and then he created the D3 toolkit, which is a JavaScript-based open-source project for creating very appealing data visualization. D3 is an acronym for “Data-Driven Documents,” and its homepage is here:

*<http://mbostock.github.com/d3/>*

Although D3 can be used for practically any type of data visualization, common use-cases include rendering maps, geographic-related data, economic data (such as employment figures) in conjunction with various locales, and medical data (diabetes seems to be very popular).

In December of 2011, D3 was named the data-visualization project of the year (by *Flowing Data!*), which is not surprising when you see the functionality that is available in D3.

D3 provides a layer of abstraction that generates underlying Scalable Vector Graphics (SVG) code. D3 enables you to create a surprisingly rich variety of data visualizations. If you need to generate graphics-oriented Web pages, and you prefer to work with JavaScript instead of working with raw SVG, then you definitely ought to consider using D3. Two key aspects of D3 involve tools for reading data in multiple formats and the ability to transform the data and render the data in many forms. D3 supports the following features:

- creation of SVG-based 2D shapes
- 2D graphics and animation effects
- method chaining

D3 has an extensive collection of “helper methods,” such as `select()`, `append()`, `data()`, and `attr()`, among others. Read the online documentation about these and other D3 methods.

## D3 ON MOBILE DEVICES

---

D3 works on any device that supports JavaScript and SVG, including mobile devices such as smart phones and tablets. These devices do vary in terms of their support for SVG features. For instance, Android 3.x has some support for SVG, and currently no version of Android supports SVG filters (discussed in Chapter 3 and Chapter 5). In general, iOS devices support more SVG features than Android-based mobile devices.

If you are writing HTML Web pages for desktops as well as mobile devices, you probably need to take into account scenarios such as handling mouse-related events versus touch-related events. In particular, you ought to test multi-touch support on multiple mobile devices and operating systems to ensure that your Web pages exhibit the expected behavior.

In addition, you might encounter D3 bugs on mobile devices that are not readily apparent in HTML Web pages laptops or desktops. If you do encounter inconsistent behavior, check the issues-related link in the previous section to see if it's a known issue. If you do not find entries, it's possible that you have discovered an unreported bug in D3, in which case you can file a new issue.

## D3 BOILERPLATE

---

If you have worked with HTML5, you are probably familiar with various boilerplate toolkits that are available. In a similar spirit, you can download a D3 boilerplate toolkit (`d3.js-boilerplate`) here:

*<https://github.com/zmaril/d3.js-boilerplate>*

According to the D3 Boilerplate Website, “d3.js boilerplate is an opinionated template system designed to help you build a sophisticated data-driven document as fast as possible. By providing a full-featured template and encouraging the use of useful tools, this project aims to help developers passively and actively cut down on development time.” If you are a D3 novice, you might not be ready to use this toolkit, but it’s a good idea to be aware of its functionality.

This concludes the brief introduction to D3. The next section of this chapter introduces you to the concept of method chaining to facilitate the discussion of subsequent code samples.

## METHOD CHAINING IN D3

---

Practically every code sample in this book (and almost all the online code samples in various forums) use method chaining, so it’s worth your time to understand method chaining before delving into the code samples.

The key idea to remember is that a D3 search actually returns a result set that is the set of elements that match the selection criteria. You can then apply an action to that set of elements. For example, you can find all the paragraphs in an HTML Web page and then set their text in red. Here is an example (taken from Listing 1.1) that uses the `d3.selectAll()` method to select all the HTML `<p>` elements in an HTML Web page and then invokes the `style()` method to set the color of the text in those paragraphs in red:

```
d3.selectAll("p").style("color", "red");
```

Returning to our previous discussion, after applying an action to a set of elements, a new set of elements is returned. In fact, you can apply a second action to that modified set, which returns yet another set. This process of applying multiple methods to a set is called method chaining, and the good news is that you can chain together as many function invocations as you wish. Method chaining enables you to write very compact yet powerful code, as you will see in the code examples in this chapter.

## THE D3 METHODS `SELECT()` AND `SELECTALL()`

---

D3 supports various selection-based methods that return arrays of arrays of elements to maintain the hierarchical structure of subselections. D3 also binds additional methods to the array of selected elements thereby enabling you to perform operations on those elements.

D3 provides the method `selectAll()` that you saw in the previous section and the method `select()`. Both methods accept selector strings, and both are used for selecting elements. *The `select()` method selects only the first matching element, whereas the `selectAll()` method selects all matching elements*

(*in document traversal order*). Due to space constraints, this chapter covers a modest subset of the D3 selection-based methods, but you can find a complete list of methods here:

*<https://github.com/mbostock/d3/wiki/Selections>*

## SPECIFYING UTF-8 IN HTML5 WEB PAGES WITH D3

All versions of D3 require UTF-8, and failing to specify UTF-8 can cause HTML5 Web pages to behave unpredictably (depending on the browser). You can ensure that your HTML5 Web pages with D3 code will work correctly by including the following snippet immediately after the `<head>` element:

```
<meta charset="utf-8" />
```

In particular, the preceding tag will ensure that your HTML5 Web pages with D3 and Unicode characters will work correctly. You will see an example of D3 with Unicode characters in Chapter 4.

## CREATING NEW HTML ELEMENTS

The code sample in this section uses method chaining, the D3 `.select()` method, and the `d3.append()` method to modify an HTML Web page. This simple example shows you how to use these two useful D3 methods.

Listing 1.1 displays the contents of `AppendElement1.html` that illustrates how to add an HTML `<p>` element to an HTML Web page.

### LISTING 1.1 *AppendElement1.html*

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8" />
    <title>Append Elements</title>
    <script src="d3.js"></script>
  </head>

  <body>
    <script>
      d3.select("body").append("p").text("Hello1 D3");
      d3.select("body").append("p").text("Hello2 D3");
      d3.select("body").append("p").text("Hello3 D3");
      d3.select("body").append("p").text("Hello4 D3");

      d3.selectAll("p").style("color", "red");
    </script>
  </body>
</html>
```

Listing 1.1 starts by referencing the D3 JavaScript file `d3.js`. Next, the `<script>` element appends four HTML `<p>` elements to the `<body>`

element using the `d3.select()` method. Finally, Listing 1.1 changes the color of all four HTML `<p>` elements to red with this code snippet:

```
d3.selectAll("p").style("color", "red");
```

Incidentally, if you want to alternate the colors in the four HTML `<p>` elements, insert the following code in Listing 1.1:

```
d3.selectAll("p").style("color", function(d, i) {
  return i % 2 ? "#f00" : "#eee";
});
```

The preceding code snippet uses a ternary operator to return the color `#f00` for even-numbered HTML `<p>` elements and `#eee` for odd-numbered HTML `<p>` elements.


**NOTE** *Some older browsers run JavaScript code before the Document Object Model (DOM) is available, in which case you can either use `window.onload()` to ensure that this does not happen, or you can insert an empty `<div></div>` element immediately after the `<body>` element and change the occurrences of `d3.select("body")` to `d3.select("div")`. Figure 1.1 displays the graphics image that is rendered by the code in the HTML Web page in Listing 1.1.*

## THE MOST COMMON IDIOM IN D3

The most common idiom in D3 (TMCHID3) for programmatically creating new DOM elements uses the following type of construct (which, of course, involves method chaining):

```
var theData = [1,2,3,4,5];

var paras = d3.select("body")
  .selectAll("p")
  .data(theData)
  .enter()
  .append("p")
  .text("D3 ");
```



Hello1 D3

Hello2 D3

Hello3 D3

Hello4 D3

**FIGURE 1.1** Dynamically Appending `<p>` Elements Using D3.



Here is how to read the code in the preceding code block, starting from the definition of the `paras` variable:

Step 1: Start by selecting the `<body>` element of the current HTML Web page (using the `select()` method).

Step 2: Return the result set of all the child `<p>` elements using the `selectAll()` method (if there are no child `<p>` elements, the returned set is a set of length zero).

Step 3: Iterate or loop through the numbers in the JavaScript array `theData` to create a new HTML `<p>` element whose text value is the string `D3`.

Step 4: After each iteration in Step 3, append the newly created `<p>` element to the result set in Step 2.

---

**NOTE** *The `d3.selectAll()` method always returns a result set, which can be an empty set (and therefore, this method never returns a null or undefined value).*

When the preceding code snippet has completed, the JavaScript variable `paras` will consist of five new HTML `<p>` elements. If you understand this sequence of events, you are ready for the code sample in the next section. If you do not understand, then continue to the next section and launch the HTML Web page in a browser to convince yourself that the preceding explanation is correct.

The acronym `TMCIID3` is a convenient way to refer to the D3 code snippet that was discussed in this section, and you will see this acronym used in the code samples throughout this book.

## BINDING DATA TO DOCUMENT-OBJECT-MODEL ELEMENTS

Now that you understand method chaining and how to use the most common idiom in D3, you are ready to see how to perform both in an HTML Web page.

Listing 1.2 displays the contents of `Binding1.html` that illustrates how to combine JavaScript variables with the D3 methods `.data()` and `.text()` to append a set of HTML `<p>` elements to an HTML Web page.

### LISTING 1.2 *Binding1.html*

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8" />
    <title>Appending Sets of Elements</title>
    <script src="d3.js"></script>
  </head>
```

```

<body>
  <script>
    var theData = [1,2,3,4,5];

    var paras = d3.select("body")
      .selectAll("p")
      .data(theData)
      .enter()
      .append("p")
      .text("D3 ");

    d3.select("body").append("paras");
  </script>
</body>
</html>

```

Listing 1.2 starts by referencing the D3 JavaScript file; the code in the `<script>` element has already been discussed in the preceding section. The only new code in Listing 1.2 is the following code snippet:

```
d3.select("body").append("paras");
```

The preceding code snippet appends the contents of the `paras` variable, which consists of five new HTML `<p>` elements, to the existing HTML `<p>` elements (if there are any) that are child elements of the `<body>` element.

**NOTE** *Because every HTML Web page in this book starts with the same boilerplate code in the HTML `<head>` element, we will omit this duplicated description for the rest of the book.*

Figure 1.2 displays the result of rendering the HTML Web page in Listing 1.2 in a Web browser.



**FIGURE 1.2** Using TMCID3 to Generate `<p>` Elements in a Web Page.

## GENERATING TEXT STRINGS

The code sample in the previous section simply generated a set of text strings with the same text. In this section, you will see how to generate a set of text strings that contain the numbers in a JavaScript array.

Listing 1.3 displays the contents of `GenerateText1.html` that illustrate how to iterate through a JavaScript array (containing numbers) and render text strings with the values in the array.

### LISTING 1.3 *GenerateText1.html*

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Iterating Through Arrays</title>
    <script src="d3.min.js"></script>
  </head>

  <body>
    <script>
      var dataValues1 = [50, 100, 250, 150, 300];

      d3.select("body")
        .selectAll("p")
        .data(dataValues1)
        .enter()
        .append("p")
        .text(function(d) { return "Paragraph Number: "+d; })
        .style("font-size", "16px")
        .style("color", "blue");
    </script>
  </body>
</html>
```

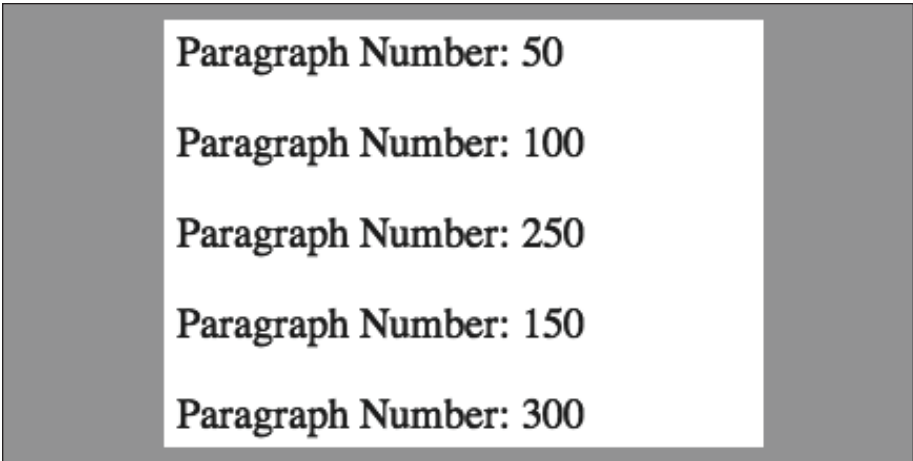
Listing 1.3 contains a `<script>` element that initializes a JavaScript variable, `dataValues1`, followed by `TMCIID3` to create and append a set of new HTML `<p>` elements to the existing HTML Web page. The only new construct is the use of a function, as shown in the following code snippet:

```
.text(function(d) { return "Paragraph Number: "+d; })
```

When you define a function in `TMCIID3`, `D3` understands that it must populate the variable `d` with the value of the current iteration through the numbers in the JavaScript array `dataValues1`.

You can use any legitimate name that you want in the preceding function, but perhaps it helps to think of the variable `d` as datum, or a single piece of data (such as a number in an array). Later, you will see functions that specify a datum and an index using the following syntax:

```
.text(function(d, i) { return d[i]; })
```



**FIGURE 1.3** Using an Array to Generate `<p>` Elements With Simple Styling Effects. The next section shows you how to leverage what you have learned about D3 to render various 2D shapes in an HTML Web page.

Figure 1.3 displays the result of rendering the HTML Web page in Listing 1.3 in a Web browser.

## CREATING SIMPLE TWO-DIMENSIONAL SHAPES

This section contains a code sample that shows you how to create simple 2D shapes in D3. The D3 code specifies attributes that are the same as the SVG-based attributes for each 2D shape. For example, an ellipse is defined in terms of its center point (`cx`, `cy`), its major axis `rx`, and its minor axis `ry`. Similar comments apply for creating a rectangle (`x`, `y`, `width`, and `height` attributes) and for creating a line segment (`(x1,y1)` and `(x2,y2)` as the coordinates of the two endpoints of the line segment). In fact, Listing 1.4 is nothing more than using `TMCIID3` and the `D3.attr()` method to set the attributes of various 2D shapes. Note that in Chapter 5 you will learn about the SVG attributes for numerous 2D shapes.

Listing 1.4 displays the contents of `SimpleShapes1.html` that illustrates how to create a circle, an ellipse, a rectangle, and a line segment in D3.

### **LISTING 1.4** *SimpleShapes1.html*

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8" />
  <title>Create Simple 2D Shapes</title>
  <script src="d3.min.js"></script>
</head>

<body>
  <script>
    var width = 600, height = 400;
```

```

// circle and ellipse attributes
var cx = 50,    cy = 80,    radius1 = 40,
    ex = 250,  ey = 80,    radius2 = 80;

// rectangle attributes
var rectX = 20, rectY = 200;
var rWidth = 100, rHeight = 50;

// line segment attributes
var x1=150,y1=150,x2=300,y2=250,lineWidth=4;

var colors = ["red", "blue", "green"];

// create an SVG element
var svg = d3.select("body")
    .append("svg")
    .attr("width", width)
    .attr("height", height);

// append a circle
svg.append("circle")
    .attr("cx", cx)
    .attr("cy", cy)
    .attr("r", radius1)
    .attr("fill", colors[0]);

// append an ellipse
svg.append("ellipse")
    .attr("cx", ex)
    .attr("cy", ey)
    .attr("rx", radius2)
    .attr("ry", radius1)
    .attr("fill", colors[1]);

// append a rectangle
svg.append("rect")
    .attr("x", rectX)
    .attr("y", rectY)
    .attr("width", rWidth)
    .attr("height", rHeight)
    .attr("fill", colors[2]);

// append a line segment
svg.append("line")
    .attr("x1", x1)
    .attr("y1", y1)
    .attr("x2", x2)
    .attr("y2", y2)
    .attr("stroke-width", lineWidth)
    .attr("stroke", colors[0]);
</script>
</body>
</html>

```

Listing 1.4 contains a `<script>` element that creates multiple SVG elements and uses the D3 `.attr()` method to set the value of the attributes of

each SVG element. When you launch the HTML Web page in Listing 1.4, D3 appends the following code block to the existing HTML Web page:

```
<svg width="600" height="400">
  <circle cx="50" cy="80" r="40" fill="red"></circle>
  <ellipse cx="250" cy="80" rx="80" ry="40" fill="blue"></ellipse>
  <rect x="20" y="200" width="100" height="50" fill="green"></rect>
  <line x1="150" y1="150" x2="300" y2="250"
        stroke-width="4" stroke="red"></line>
</svg>
```

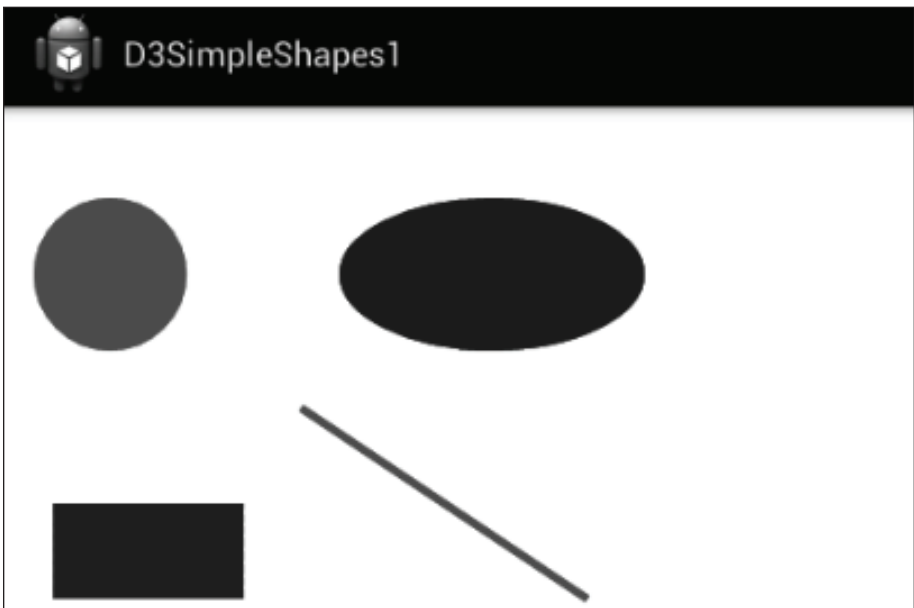
Compare the code in Listing 1.4 with the preceding code block to verify that the preceding SVG elements correspond to the code in Listing 1.4. You can use an alternate coding style that defines multiple JavaScript variables, as shown in the following code block:

```
var theBody = d3.select("body");

var theSVG = theBody.append("svg")
  .attr("width", 100)
  .attr("height", 100);

var circleSelection = theSVG.append("circle")
  .attr("cx", 50)
  .attr("cy", 50)
  .attr("r", 30)
  .style("fill", "red");
```

Figure 1.4 displays the graphics image that is rendered by the code in the HTML Web page in Listing 1.4.



**FIGURE 1.4** D3 Code for a Circle, an Ellipse, a Rectangle, and a Line Segment.

## BEZIER CURVES AND TEXT

Listing 1.5 displays the contents of `BezierCurvesAndText1.html` that illustrates how to use D3 to render a quadratic Bezier curve, a cubic Bezier curve, and text strings that follow the path of the Bezier curves.

### LISTING 1.5 *BezierCurvesAndText1.html*

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8" />
    <title>Bezier Curves and Text</title>
    <script src="d3.min.js"></script>
  </head>

  <body>
    <script>
      var width = 600, height = 400, opacity=0.5;
      var cubicPath  = "M20,20 C300,200 100,500 400,100";
      var quadPath   = "M200,20 Q100,300 500,100";

      var cValues     = "M20,20 C300,200 100,500 400,100";
      var qValues     = "M200,20 Q100,300 500,100";

      var fontSizeC   = "24";
      var fontSizeQ   = "18";

      var textC =
        "Sample Text that follows a path of a cubic Bezier curve";

      var textQ =
        "Sample Text that follows a path of a quadratic Bezier curve";

      var fillColors = ["red", "blue", "green", "yellow"];

      // create an SVG container...
      var svgContainer = d3.select("body").append("svg")
        .attr("width", width)
        .attr("height", height);

      var defs        = svgContainer
        .append("svg:defs");

      var patternC = defs.append("svg:path")
        .attr("id", "pathInfoC")
        .attr("d", cValues);

      var patternQ = defs.append("svg:path")
        .attr("id", "pathInfoQ")
        .attr("d", qValues);

      // now add the 'g' element...
      var g1 = svgContainer.append("svg:g");
```

```

// create a cubic Bezier curve...
var bezierC = gl.append("path")
    .attr("d", cubicPath)
    .attr("fill", fillColors[0])
    .attr("stroke", "blue")
    .attr("stroke-width", 2);

// text following a cubic Bezier curve...
var textC = gl.append("text")
    .attr("id", "textStyleC")
    .attr("stroke", "blue")
    .attr("fill", fillColors[1])
    .attr("stroke-width", 2)
    .append("textPath")
    .attr("font-size", fontSizeC)
    .attr("xlink:href", "#pathInfoC")
    .text(textC);

// create a quadratic Bezier curve...
var bezierQ = gl.append("path")
    .attr("d", quadPath)
    .attr("fill", fillColors[1])
    .attr("opacity", opacity)
    .attr("stroke", "blue")
    .attr("stroke-width", 2);

// text following a cubic Bezier curve...
var textQ = gl.append("text")
    .attr("id", "textStyleQ")
    .attr("stroke", fillColors[3])
    .attr("stroke-width", 2)
    .append("textPath")
    .attr("font-size", fontSizeQ)
    .attr("xlink:href", "#pathInfoQ")
    .text(textQ);

</script>
</body>
</html>

```

Listing 1.5 contains the usual boilerplate code and a `<script>` element that defines a quadratic Bezier curve and a cubic Bezier curve. The JavaScript variables `bezierC`, `bezierQ`, `textC`, and `textQ` are set to the values for a cubic Bezier curve, a quadratic Bezier curve, the text for the cubic Bezier curve, and the text for the quadratic Bezier curve, respectively. These variables are used in the D3 code that creates a `<path>` element, which is how you specify quadratic and cubic Bezier curves in SVG.

As you can see, most of the code in Listing 1.5 does two things: it creates SVG elements with the D3 `.append()` method, and it then sets the required attributes with the D3 `.attr()` method.

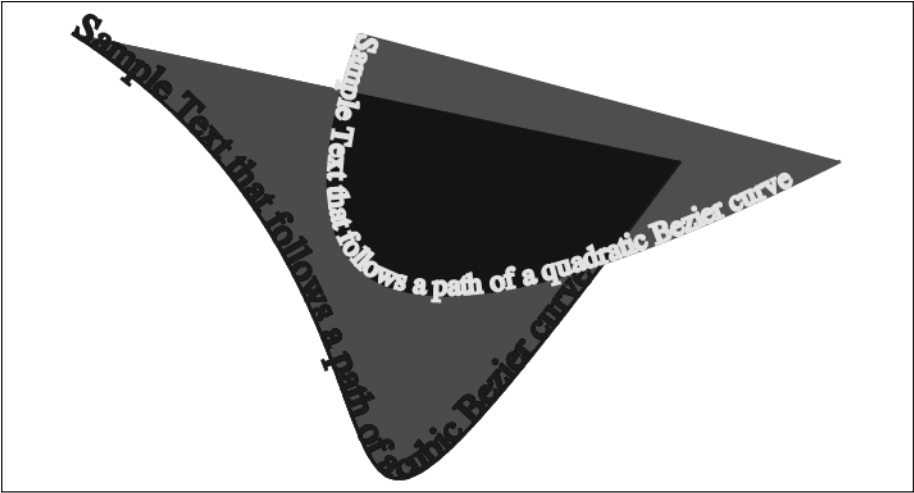
Moreover, the definitions for the cubic Bezier curve and the quadratic Bezier curve consist of a string of values, as shown here:

```

var cubicPath = "M20,20 C300,200 100,500 400,100";
var quadPath  = "M200,20 Q100,300 500,100";

```





**FIGURE 1.5** Generating Text Along Two Bezier Curves in D3.

In fact, if you want to use standard SVG code instead of D3, you could literally copy and paste the preceding strings as the value for the `d` attribute in the SVG `<path>` element.

Figure 1.5 displays the graphics image that is rendered by the code in the HTML Web page in Listing 1.5.

## TWO-DIMENSIONAL TRANSFORMS

D3 provides support for four 2D transforms: rotate, scale, skew, and translate. You can apply transforms to 2D shapes using the D3 `.attr()` method, as shown in the following examples:

```
.attr("transform", "translate("+transX+", "+transY+")");
.attr("transform", "rotate("+rotateX+")");
.attr("transform", "scale("+scaleX+", "+scaleY+")");
.attr("transform", "skewX("+skewX+")");
```



The CD contains the HTML Web page `Transforms1.html` that fully illustrates how to create four 2D transform effects.

## A DIGRESSION: SCALING ARRAYS OF NUMBERS TO DIFFERENT RANGES

This section covers the `d3.range()` method for determining the range of a set of numbers followed by the `d3.scale()` function for scaling a set of numbers. The rationale for including this section here is that the `d3.range()` method is used in the gradient-related code samples that you will see later in this chapter. This method is both easy to use and straightforward to understand, and you will see this method in many code samples in this book.

When you work with D3 functions that scale the values in an array, keep in mind that the domain specifies the input values that you provide, and the range refers to the target values that are calculated based on the domain values.

The simplest use of the `d3.range()` method is to generate a list of integers. For example, `d3.range(15)` generates the integers between 0 and 14. You can verify this fact by including the following code snippet in a D3-based HTML Web page:

```
console.log("range from 0 to 14: "+d3.range(15))
```

When you open the Web Inspector (see comments in the Preface) you will see the following:

```
Numbers from 0 to 14: 0,1,2,3,4,5,6,7,8,9,10,11,12,13,14
```

Another use for the `d3.range()` method enables you to scale the values in a JavaScript array by mapping them to a different range of values. This functionality is very useful whenever you need to scale the elements in a bar chart or graph so the graphics output (such as the individual bar elements) fits the dimensions of the screen where you are rendering the chart or graph.

Suppose that you have the following set of numbers in a JavaScript array:

```
var dataValues = [10, 20, 30, 40, 50];
```

You can scale them to the range `[0, 10]` with the following code snippet:

```
x = d3.scale.linear().domain([10,50]).range([0,10]);
```

Although the preceding code snippet is correct, there are two limitations. First, the minimum and maximum values of the JavaScript array `dataValues` are hard-coded with the values 10 and 50. Second, the range of values is also hard-coded in the code.

The following code snippet is a better way to scale a set of numbers:

```
var dataValues = [10, 20, 30, 40, 50], left=0, right=10;
var xScale = d3.scale.linear()
    .domain([d3.min(dataValues), d3.max(dataValues)])
    .range([left, right]);
```

Notice that all the hard-coded values have been replaced by JavaScript variables. Although we have not discussed the `d3.min()` and `d3.max()` methods, they return the minimum and maximum values, respectively, in a JavaScript array of numbers.

The advantage of the preceding code block is that it works correctly for *any* JavaScript array of numbers. However, you do need to make a manual change to the range values if you want to use a different range.

---

**NOTE** *You will probably indent the range of values so there is padding on the left and right of your charts and graphs to avoid inadvertently clipping portions of data from the visual display.*

For example, if you want to render a scatter plot in the horizontal range of `[0, 600]`, and you also want to indent by 20 on the left and on the right of the chart, you can use something like the following code snippet:

```
var pad=20;
xScale = d3.scale.linear()
    .domain([d3.min(dataValues), d3.max(dataValues)])
    .range([left+pad, right-pad]);
```

The preceding code block is for scaling numbers along the horizontal axis, and the same type of code works for scaling numbers along the vertical axis, as shown here:

```
yScale = d3.scale.linear()
    .domain([d3.min(dataValues), d3.max(dataValues)])
    .range([pad, height-pad]);
```

---

**NOTE**

*The vertical axis is positive in the top-to-bottom direction, and if you want to reverse the polarity of the vertical axis, simply reverse the two numbers in the `d3.range()` method in the preceding definition for `yScale`.*

Other convenient array-related functions are also available in D3. For example, if you want to reverse the order of the integers between 1 and 4 inclusive, you can use this code snippet:

```
var range1 = d3.range(1,5).reverse();
```

The next chapter shows more examples of JavaScript arrays in D3 in greater detail.

## TWEENING IN D3

---

The term *tweening* refers to the process of calculating the numbers between a start value and an end value. The tweened values are of the same type as the start and end values (tweening a pair of numbers produces a set of numbers, and tweening a pair of colors produces a set of colors). There are different formulas for different tweening effects. By way of analogy, think of accelerating in a car: a smooth acceleration from 0 to 50 is one type of tweening effect that produces a smooth experience. On the other hand, a rollercoaster has one or more intervals involving slow-fast-slow acceleration, which provides another type of tweening effect.

D3 provides support for three tweening methods: `styleTween()`, `attrTween()`, and `tween()`. An example of using `styleTween()` is here:

```
d3.select("body").transition()
    .styleTween("color", function() {
        return d3.interpolate("green", "red");
    });
```

A fun tip: you can use colors in addition to numbers in the `d3.range()` function. For example, the following code snippet is valid in D3:

```
var colorScale = d3.scale.linear()
    .domain([0,100])
    .range(['red', 'blue']);

...
.attr('fill', function(d) {
    return colorScale[d];
})
...

```

The preceding code block sets the color of a shape (not shown here) to a value that is a linear interpolation between `red` and `blue`. Moreover, you can specify hexadecimal values, (R,G,B) values, and HSL values, in addition to common color names, which is a very nice feature of D3.

You will see many code samples in this book that use this type of code, so you will have plenty of opportunity to become more comfortable with this coding technique in D3.

## FORMATTING NUMBERS

---

D3 supports various formats for numbers. As a simple example, you can insert a comma “,” in a number. For example, if you want to render 1234000 as 1,234,000 you can use the following code snippet:

```
var x1 = 1234000;
var x2 = d3.format(",")(x1)
d3.format(".3s");

```

If you want to display 1234000 as 1.234M, use the following code snippet:

```
var x1 = 1234000;
var x3 = d3.format(".4s")(x1);

```

More information about D3 formatting is here:

[https://github.com/mbostock/d3/wiki/Formatting#wiki-d3\\_format](https://github.com/mbostock/d3/wiki/Formatting#wiki-d3_format)

## WORKING WITH GRADIENTS

---

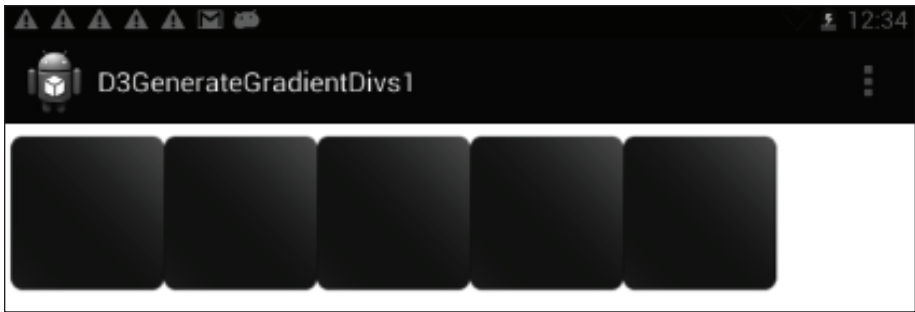
SVG supports linear gradients and radial gradients, and therefore, you can also render both of these gradients in D3. The next two code samples show you how to render 2D shapes with linear gradients and radial gradients in D3.

## LINEAR GRADIENTS

---



D3 enables you to define linear gradients in a straightforward manner. The HTML Web page `LinearGradient1.html` on the CD illustrates how to



**FIGURE 1.6** A Linear Gradient in D3.

create linear gradients in D3. The definition of a linear gradient is shown in the following code block:

```
var gradient = svg.append("svg:defs")
    .append("svg:linearGradient")
    .attr("id", "gradient")
    .attr("x1", "0%")
    .attr("y1", "0%")
    .attr("x2", "100%")
    .attr("y2", "100%")
    .attr("spreadMethod", "pad");

gradient.append("svg:stop")
    .attr("offset", "0%")
    .attr("stop-color", "#0c0")
    .attr("stop-opacity", 1);

gradient.append("svg:stop")
    .attr("offset", "100%")
    .attr("stop-color", "#c00")
    .attr("stop-opacity", 1);
```

The preceding code block starts with a code snippet by creating an SVG `<defs>` element that is appended to an SVG `<svg>` element (not shown here). The next two code snippets define a so-called stop color that specifies the attributes of the color to render in the linear gradient. Figure 1.6 displays the graphics image that is rendered by the code in the HTML Web page `LinearGradient1.html`.

## RADIAL GRADIENTS

In addition to linear gradients, D3 enables you to define radial gradients. The HTML Web page `RadialGradient1.html` illustrates how to create radial gradients in D3. The definition of a radial gradient is very similar to a linear gradient, and the difference is shown in the following code block:

```

var gradient = svg.append("svg:defs")
    .append("svg:radialGradient")
    .attr("id", "gradient")
    .attr("x1", "0%")
    .attr("y1", "0%")
    .attr("x2", "100%")
    .attr("y2", "100%");

gradient.append("svg:stop")
    .attr("offset", "0%")
    .attr("stop-color", "#f00")
    .attr("stop-opacity", 1.0);
// add other stop colors as needed

```



Read the entire source code on the CD to see the omitted details.

## ADDING HTML <DIV> ELEMENTS WITH GRADIENT EFFECTS

If you have worked with CSS3 graphics, the CSS3 selectors in Listing 1.6 will be familiar to you. If you are new to CSS3, this code sample gives you a preview of how easily you can combine CSS3 with D3 in an HTML Web page. CSS3 graphics and animation are the subject of Chapter 6, so you can revisit this code sample after you have finished reading Chapter 6.

Listing 1.6 displays the contents of `GenerateGradientDivs1.html` that illustrates how to iterate through an array and render text strings that are displayed with CSS3 gradients.

### **LISTING 1.6** *GenerateGradientDivs1.html*

```

<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <title>Creating Gradients </title>
  <script src="d3.min.js"></script>

  <style>
    .gradient {
      display: inline-block;
      top: 20px;
      width: 100px;
      height: 100px;

      background-image: -webkit-gradient(linear,
        100% 0%, 0% 100%,
        from(#f00), to(#00f));
      background-image: -gradient(linear,
        100% 0%, 0% 100%,
        from(#f00), to(#00f));

      border-radius: 8px;
    }
  </style>
</head>

```

```

<body>
  <script>
    var dataValues1 = [0, 150, 300, 450, 600];

    d3.select("body")
      .selectAll("p")
      .data(dataValues1)
      .enter()
      .append("div")
      .attr("left", function(d) {
        return d+"px";
      })
      .attr("class", "gradient");
  </script>
</body>
</html>

```

Listing 1.6 contains the usual boilerplate code, and the `<script>` element uses `TMCIID3` to create and append a set of HTML `<div>` elements to the existing HTML Web page. There are two new things to notice about this code. First, there is a function definition that uses the D3 `.attr()` method, as shown here:

```

.attr("left", function(d) {
  return d+"px";
})

```

The preceding code block uses each and every value in the JavaScript array `dataValues1` (remember that D3 is iterating through an array behind the scenes) to set the pixel value of the CSS `left` property of each new HTML `<div>` element. Second, the following code snippet sets the CSS class property to `.gradient` (which is defined as a selector in the `<style>` element near the top of Listing 1.6):

```

.attr("class", "gradient");

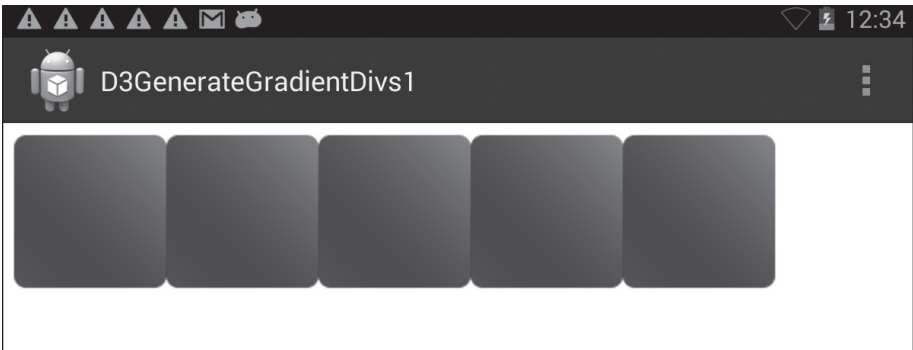
```

Launch the code in Listing 1.4 and view the source code to verify the previous statements and to familiarize yourself with this technique for using D3 to create gradient effects in HTML Web pages. Figure 1.7 displays the graphics image that is rendered by the code in the HTML Web page in Listing 1.6.

## WORKING WITH PORTABLE NETWORK GRAPHICS FILES

You can use D3 to specify portable network graphics (PNG) files in an HTML Web page. For example, the HTML Web page `Images1.html` on the CD illustrates how to render three PNG files. You need to specify values for the attributes `x`, `y`, `width`, `height`, and the name of the PNG file. The key idea is shown in the following code block:





**FIGURE 1.7** Applying Gradients to <div> Elements in D3.

```
var imageWidth=100, imageHeight=100;
var dataValues = [[0,0], [100, 100], [200, 200]];
var images = ["sample1.png", "sample2.png", "sample3.png"];

// code omitted for brevity
.append("svg:image")
.attr("xlink:href", function(d,i) {
    return images[i];
})
.attr("width", imageWidth)
.attr("height", imageHeight)
.attr("x", function(d) { return d[0]; })
.attr("y", function(d) { return d[1]; })
```



Read the entire source code available on the CD to see the omitted details.

## D3 APPLICATION PROGRAMING INTERFACE REFERENCE

If you are impatient and you want to dive into the D3 APIs, they are listed here:

<https://github.com/mbostock/d3/wiki/API-Reference>

The D3 APIs are listed in various categories, including: Ajax, arrays, axes, chord, cluster, colors, force, geography, hierarchy, histogram, ordinals, pie, projections, scales, shapes (SVG), selections, stack, string formatting, time, transitions, tree, and treemap. In addition, known issues for D3 are listed here:

<https://github.com/mbostock/d3/issues?page=1&state=open>

## ADDITIONAL CODE SAMPLES ON THE CD



The CD contains additional HTML Web pages that use D3. The HTML Web page `Polygon1.html` illustrates how to use D3 to render a polygon. The HTML Web page `EllipticArcs1.html` renders an elliptic arc, and



the HTML Web page `BezierCurves1.html` renders a quadratic Bezier curve and a cubic Bezier curve. The HTML Web page `NestedRectangles1.html` renders a set of nested rectangles with D3.

The HTML Web page `MultiQBezierScaledGlasses1.html` shows you how to render a set of eye glasses with D3. The HTML Web page `NoU-LeftTurn1.html` shows you how to render a traffic sign with D3. The HTML Web page `PartialBlueSphereCB5.html` shows you how to render a traffic sign with D3. Note that the preceding three code samples use an SVG `<pattern>` element and also an SVG `<use>` element, neither of which are discussed in this chapter.

The HTML Web page `CardioidEllipses1Grad2.html` renders a set of gradient ellipses, each of which follow the path of a Cardioid polar equation, and the HTML Web page `Transforms1.html` illustrates how to create four 2D transform effects. The HTML Web pages `Cube1.html` and `Pyramid1.html` demonstrate how to render a cube and a pyramid, respectively, in D3. The HTML Web pages `CubeLGrad1.html` and `PyramidRGrad1.html` demonstrate how to render a cube and a pyramid with a linear gradient and a radial gradient in D3.

The HTML Web page `Cone1.html` shows you how to render a cone in D3. The HTML Web pages `GridCubes1.html` shows you how to render a grid-like set of cubes. In Chapter 2, this code sample is enhanced with animation effects and with event handlers for mouse events.

The HTML Web pages `Cone2.html`, `Pyramid2.html`, `Pyramid2.html` and `CardioidEllipses1GradUse1.html` have one thing in common: they all define an SVG `svg:use` element to create multiple scaled copies of a base graphics image that is defined in a lower-numbered code sample (e.g., `Cone2.html` contains multiple copies of a base cone that is rendered in `Cone1.html`, and so forth). Finally, an open-source project with more than 1,000 D3-based code samples (many of which are similar variants) using various polar equations is here:

*<http://code.google.com/p/d3-graphics>*

## SUMMARY

This chapter started with an introduction to D3, followed by examples of using D3 methods to render simple 2D shapes, such as circles, line segments, and rectangles. The CD also contains code samples for rendering polygons, ellipses, elliptic arcs, and Bezier curves. In addition, you learned about `TMCIID3`, which is an extremely useful code construct that is ubiquitous in HTML Web pages that use D3. In particular, you learned about the following D3 methods:

- `select()`
- `selectAll()`



- `append()`
- `enter()`
- `domain()`
- `range()`
- `styleTween()`



Finally, you learned how to render linear gradients and radial gradients, along with examples of rendering abstract graphics on the CD. The next chapter explores arrays, mouse events, and animation effects with D3.



## *ARRAYS, MOUSE EVENTS, AND ANIMATION EFFECTS*

**T**his chapter covers a diverse set of topics including JavaScript arrays, mouse events, and simple animation effects. Most (if not all) of the topics in this chapter are used in bar charts and graphs that are discussed in Chapter 3 and Chapter 4. For instance, D3 uses JavaScript arrays to represent data, so it's obviously important to know how to deal with arrays. In addition, you can use mouse events to create an interactive user experience and to display more detailed information whenever users hover over different parts of an HTML Web page. Finally, animation effects can create even more vivid effects in your data visualization, and also create a time-based effect for datasets.

If you are a D3 novice, it's probably better for you to read (or at least skim) this chapter before reading the other chapters in this book. Doing so makes it easier for you to quickly refer back to this chapter when you see the associated material used in subsequent chapters. For example, this chapter discusses D3 scaling functions, which are used in practically every chart and graph in this book. If you have already read this chapter then you can quickly refer back to the section about scaling functions if you need to; similar comments apply to the other topics in this chapter.

With the preceding point in mind, the first part of this chapter shows you how to work with multidimensional arrays in D3, which is useful for representing complex data sets. The second part of this chapter shows you how to use D3 with mouse-related events, which are handy for creating interactive data-visualization Websites.

The final portion of this chapter contains code samples for creating animation effects. D3 enables you to create fine-grained animation effects more easily than pure SVG, thereby enabling you to enhance your Web pages with subtle and pleasing visual effects.

One more point: The only way to capture the full visual effect of the Web pages that create animation effects or mouse-related resizing effects is by launching those Web pages in a browser. A series of screenshots that approximate the visual effects cannot provide a comparable substitute, so screenshots for those Web pages are not included in this chapter.

## FINDING THE MAXIMUM AND MINIMUM VALUES IN AN ARRAY

In Chapter 1 you learned how to use the `d3.domain()` and the `d3.range()` functions, both of which are used extensively in the code samples in this book. D3 also provides the functions `d3.min()` and `d3.max()`, which find the minimum and maximum value of a set of numbers. Although you can programmatically determine the maximum and minimum numbers in a JavaScript array, the D3 methods conveniently perform these calculations for you. For example, suppose that you have a set of data points whose minimum and maximum values can change dynamically. In addition, suppose that you also want the ability to map the data points to different value ranges. The following code block provides the required type of flexibility and functionality:

```
var dataValues = [10, 20, 30, 40, 50], lower=100, upper=400;

var xScale = d3.scale.ordinal()
    .domain([d3.min(dataValues), d3.max(dataValues)])
    .range([lower, upper]);
```

As you can see, the use of the `d3.min()` and `d3.max()` functions in the preceding code block ensures that you can replace the `dataValues` array with any realistic set of data values. In addition, you can change the values of `lower` and `upper` independently of the data values, which means that you can scale the vertical axis to accommodate your needs (and device types).

## WORKING WITH MULTIDIMENSIONAL ARRAYS

D3 supports several data formats (including comma-separated value [CSV]) that are discussed in Chapter 4, and you can also write D3 code that references data that is defined in multidimensional JavaScript arrays. Listing 2.1 displays the contents of `IterateArrays1.html` that illustrate how to iterate through some of the values of three arrays.

### **LISTING 2.1** *IterateArrays1.html*

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Iterating Through Arrays</title>
    <script src="d3.min.js"></script>
  </head>
```

```

<body>
<script>
  var width      = 600, height  = 400,
      sWidth     = 20,  sHeight = 20,
      offsetX    = 0,   offsetY = 0;

  var svg = d3.select("body")
    .append("svg:svg")
    .attr("width", width)
    .attr("height", height);

  var dataValues1 = [50, 100, 250, 150, 300];

  var dataValues2 = [
    [50, 20], [100, 90], [250, 50], [150, 33], [300, 95]
  ];

  var dataValues3 = [
    [50, 20, 90], [100, 90, 50], [250, 50, 20]
  ];

  d3.max(dataValues1, function(d) {
    console.log("current dataValues1 d: "+d);

    var mySquare = svg.append("rect")
      .attr("x", d)
      .attr("y", offsetY)
      .attr("width", sWidth)
      .attr("height", sHeight)
      .style("fill", "red");

    return d;
  });

  d3.max(dataValues2, function(d) {
    console.log("current dataValues2 d: "+d[0]);
    return d[0];
  });

  d3.max(dataValues3, function(d) {
    console.log("current dataValues3 d: "+d[1]);
    return d[1];
  });
</script>
</body>
</html>

```

Listing 2.1 starts with the usual boilerplate code followed by a `<script>` element that defines three JavaScript arrays, `dataValues1`, `dataValues2`, and `dataValues3`, containing one-dimensional, two-dimensional, and three-dimensional arrays, respectively. Listing 2.1 uses the values in `dataValues1` to generate a set of rectangles, which you have seen in earlier code samples.

The next code snippet displays the maximum value of the first component in the `dataValues2` array, as shown here:

```
d3.max(dataValues2, function(d) {
  console.log("current dataValues2 d: "+d[0]);
  return d[0];
});
```

Similarly, the final code snippet displays the maximum value of the second component in the `dataValues3` array. While there is nothing exceptional about this code, it shows you how to access different elements of multidimensional JavaScript arrays and how to apply D3 methods to those arrays.

In addition, Listing 2.1 shows you how to define a JavaScript function that uses D3 to generate a set of random numbers, as shown here:

```
function randomData(rangeCount, maxSize) {
  return d3.range(rangeCount).map(function(i) {
    // return a 2-dimensional array...
    return {x: i, y: maxSize*Math.random()};
  });
}

// 20 random numbers (maximum value of each is 100)
var data = randomData(20, 100);

// find the maximum value for 2d array
var maxValue2 = d3.max(data, function(d) {return d.y;});
```

Figure 2.1 is a screenshot of a Chrome browser session that displays the generated output from Listing 2.1.

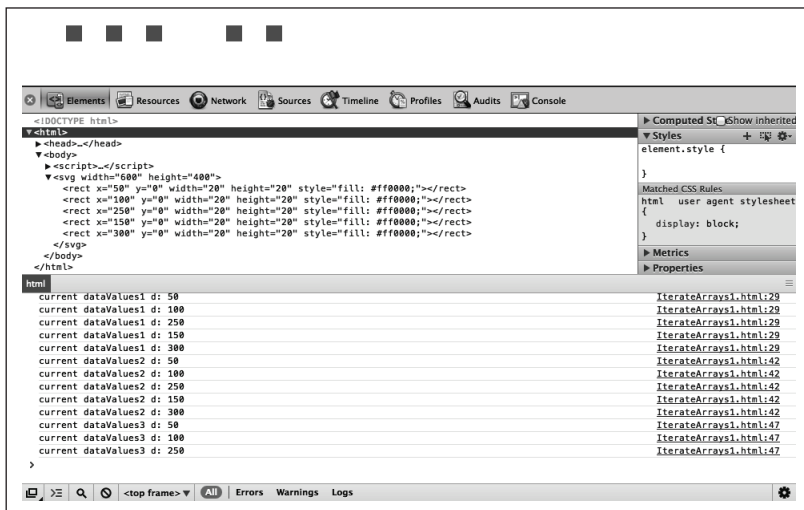


FIGURE 2.1 Output Values from the Loops in Listing 2.1.

## TWO-DIMENSIONAL ARRAYS AND SCATTER CHARTS

Now that you understand how to iterate through the values of 2D arrays, it's very easy to generate scatter charts. The idea involves rendering circles whose center point is based on the pairs of values that belong to a 2D array. Although this code sample could easily be part of Chapter 2, it's a good opportunity to leverage the concepts that you learned in the preceding section. Listing 2.2 displays the contents of `IterateArrays2.html` that illustrate how to render a scatter chart.

### **LISTING 2.2** *IterateArrays2.html*

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Iterating Through Arrays</title>
    <script src="d3.min.js"></script>
  </head>

  <body>
    <script>
      var width = 600, height = 400, radius = 8;

      var svg = d3.select("body")
        .append("svg:svg")
        .attr("width", width)
        .attr("height", height);

      var dataValues2 = [
        [50, 20], [100, 200], [250, 150], [150, 30],
        [30, 80], [200, 140], [150, 190], [350, 50]
      ];

      // render the circles...
      d3.max(dataValues2, function(d) {
        svg.append("circle")
          .attr("cx", d[0])
          .attr("cy", d[1])
          .attr("r", radius)
          .style("fill", "red");
      });

      // display the coordinates...
      svg.selectAll("text")
        .data(dataValues2)
        .enter()
        .append("text")
        .text(function(d) {
          return d[0] + "," + d[1];
        })
        .attr("x", function(d) {
          return d[0];
        })
    </script>
  </body>
</html>
```



```

        .attr("y", function(d) {
            return d[1];
        })
        .attr("font-family", "sans-serif")
        .attr("font-size", "11px")
        .attr("fill", "blue");
    </script>
</body>
</html>

```

Listing 2.2 starts with the usual boilerplate code followed by a `<script>` element that defines the JavaScript array `dataValues2` two-dimensional elements. Next, Listing 2.2 uses the values in `dataValues2` to generate a set of circles, which you have seen in a previous code sample, and sets values for the attributes `cx` and `cy` by specifying `d[0]` and `d[1]`, which you learned in the preceding section.

The next portion of code uses `TMCIID3` to render the coordinates of each point in the scatter chart. The key point to observe is the following code snippet:

```

    .text(function(d) {
        return d[0] + "," + d[1];
    })

```

The preceding code snippet returns a pair of values that represent the coordinates of each point in the scatter chart. For example, the first pair is `(50, 100)`. The location of the text is the same location as the location of its corresponding circle, which is set with the following code block:

```

    .attr("x", function(d) {
        return d[0];
    })
    .attr("y", function(d) {
        return d[1];
    })

```

Figure 2.2 is a screenshot of a Chrome browser session that displays the generated output from Listing 2.2.

## D3 DATA SCALING FUNCTIONS

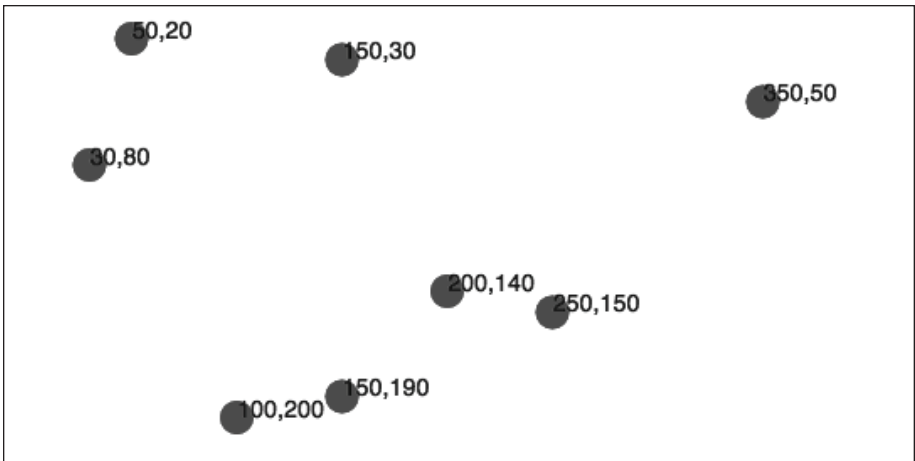
D3 provides the methods `d3.domain()` and `d3.range()` to scale data values with a linear function (as you will see shortly) along with the methods `d3.min()` and `d3.max()` that calculate the minimum value and maximum value in an array. Listing 2.3 displays the contents of `Scaling1.html` that illustrate how to scale or map a set of domain and range values.

### **LISTING 2.3** *Scaling1.html*

```

<!DOCTYPE html>
<html>
  <head>

```



**FIGURE 2.2** Rendering a Scatter Plot of Circles and Their Coordinates.

```

<meta charset="utf-8" />
<title>Scaling Effects with D3</title>
<script src="d3.min.js"></script>
</head>

<body>
  <script>
    var w = 640, h = 400,
        deltaU = 0.2, deltaV = 0.2,
        x = d3.scale.linear().domain([-1,1]).range([0,w]),
        y = d3.scale.linear().domain([0, 1]).range([0,h]);

    for(var u=-1; u<2; u += deltaU) {
      console.log("u: "+u+" x(u): "+x(u));
    }

    for(var v=0; v<2; v += deltaV) {
      console.log("v: "+v+" y(v): "+y(v));
    }
  </script>
</body>
<html>

```

Listing 2.3 starts with the usual boilerplate code followed by a `<script>` element that maps the points in `[-1, 1]` to `[0, w]` and then maps the points in `[0, 1]` to `[0, h]`. Next, there are two simple loops that iterate through both newly mapped sets of points.

Based on the linear interpolation defined in Listing 2.3, here are some sample values:

```

x(0):    320; // midpoint of [-1,1] ==> half of w
y(0.5):  200; // midpoint of [0, 1] ==> half of h
x(2):    320; // double endpoint ==> double w
y(2):    800; // double endpoint ==> double h

```

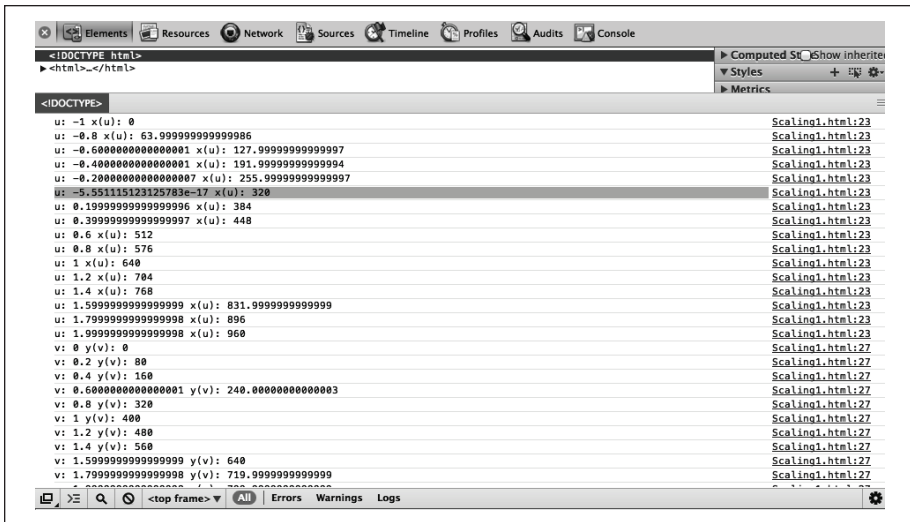


FIGURE 2.3 Output Values from the Loops in Listing 2.3.

Figure 2.3 is a screenshot of a Chrome browser session that displays the generated output from Listing 2.3.

If you want to specify horizontal and vertical indentation, you can use four JavaScript variables, or you can use the following type of code block where the JavaScript variable `m` maintains the margins in one convenient location:

```
var m = {top: 20, right: 20, bottom: 20, left: 60};

var xRange = d3.scale.linear().range([m.left, width - m.right]),
    yRange = d3.scale.linear().range([height - m.top, m.bottom]);
```

As a supplemental set of related code exercises, see if you can predict the behavior of the following code block:

```
var x1 = d3.scale.linear();
var x2 = d3.scale.linear().domain([0,10000]);
var x3 = d3.scale.linear().domain([0,10000]).range([0,100]);

var myNumbers = [100, 500, 300, 900, 2000];

// map [0,2000] to [1,100]
var x4 = d3.scale.linear()
    .domain([0,d3.max(myNumbers)])
    .range([0,100]);

// map [0,100] to [1,100]
var x5 = d3.scale.linear()
    .domain([0,d3.min(myNumbers)])
    .range([0,100]);
```

```
// what does this do?
var x6 = d3.scale.linear()
    .domain([x4, x5])
    .range([0,100]);
```

Just to be sure that you are correct, copy the preceding code block into an HTML Web page and add some `console.log()` statements. Launch your Web page and examine the Web Inspector to view the displayed information.

## OTHER D3.JS SCALING FUNCTIONS

In addition to the `Linear Scale` that we have already discussed, D3.js also supports `Quantitative Scale` and `Ordinal Scale`. In brief, `Quantitative Scales` have a continuous domain such as dates, times, and real numbers. On the other hand, `Ordinal Scales` are for discrete domains such as names, categories, and colors. Although this book does not use these scales, you can perform an Internet search to find examples of other D3 scales if you need to incorporate them into your HTML5 Web pages.

One other point to keep in mind: D3.js `Linear Scales` enable us to resize our data to fit into our predefined SVG coordinate space instead of resizing our SVG coordinate space to fit our data.

## D3 PATH DATA GENERATOR

This section shows you how to use the `D3 Path Data Generator` to create visual effects that are more complex than the code samples you saw earlier in this chapter.

Listing 2.4 displays the contents of `PathLineSegments.html` that illustrate how to render a polyline that consists of a set of line segments.

### **LISTING 2.4** *PathLineSegments1.html*

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8" />
  <title>Path-based Line Segments </title>
  <script src="d3.min.js"></script>
</head>

<body>
  <script>
    var width = 600, height = 400, radius = 5, factor=3;

    var lineData = [{ "x": 20,   "y": 50},    { "x": 200, "y": 200},
                    { "x": 240,  "y": 10},    { "x": 400, "y": 300},
                    { "x": 380,  "y": 150},    { "x": 500, "y": 250}];
```

```

// define a Path Data Generator for lines...
var lineFunction = d3.svg.line()
    .x(function(d) { return d.x; })
    .y(function(d) { return d.y; })
    .interpolate("linear");

// create an SVG container...
var svgContainer = d3.select("body").append("svg")
    .attr("width", width)
    .attr("height", height);

// add the path element (with line segments)...
var lineGraph = svgContainer.append("path")
    .attr("d", lineFunction(lineData))
    .attr("stroke", "blue")
    .attr("stroke-width", 2)
    .attr("fill", "none");

// add circles at each vertex...
d3.max(lineData, function(d) {
    var circles = svgContainer
        .append("circle")
        .data(lineData)
        .attr("cx", d.x)
        .attr("cy", d.y)
        .attr("r", radius)
        .attr("fill", "red")
        .on("mouseover", function() {
            var r = d3.select(this).attr("r");
            d3.select(this).attr("r", factor*r);
        })
        .on("mouseout", function() {
            var r = d3.select(this).attr("r");
            d3.select(this).attr("r", r/factor);
        })
    });
</script>
</body>
</html>

```

Listing 2.4 starts with the usual boilerplate code followed by a `<script>` element that defines a JavaScript array `lineData` with data based in JavaScript Object Notation (JSON) that specifies the x- and y-coordinates for a set of points in a so-called scatter chart. If you are unfamiliar with JSON, please read the appropriate section in Chapter 4.

The key idea is to define a line function that uses linear interpolation to display the points that are defined in the array `lineData`, as shown here:

```

var lineFunction = d3.svg.line()
    .x(function(d) { return d.x; })
    .y(function(d) { return d.y; })
    .interpolate("linear");

```

Notice that previous code samples in the chapter used `d[0]` and `d[1]` to reference the first and second coordinates of unnamed data values in two-dimensional JavaScript arrays. On the other hand, the preceding code snippet uses keyed data values `d.x` and `d.y` because that is how the first and second coordinates are identified in the array `lineData`.

Next, Listing 2.4 appends an SVG `<svg>` element to the HTML Web page and renders a set of line segments based on the values in the variable `lineData`. The key point to observe is the portion in bold that is shown in the following code snippet:

```
var lineGraph = svgContainer.append("path")
    .attr("d", lineFunction(lineData))
    .attr("stroke", "blue")
    .attr("stroke-width", 2)
    .attr("fill", "none");
```

The preceding code snippet does a lot of work behind the scenes: it uses the definitions in the `lineFunction` variable to create an SVG `<path>` element that consists of a set of line segments whose vertices are from the data points in the variable `lineData` (re-read this sentence to absorb all of the information).

The final portion of code in Listing 2.4 creates a set of SVG `<circle>` elements using code that you have seen in previous code samples.

Incidentally, D3.js provides eleven different types of line interpolations for the `d3.svg.line()` function, and you can perform an Internet search for code samples that use those interpolations if you are interested in using them in your HTML Web pages.

## WHAT ABOUT THIS, \$THIS, AND \$(THIS)?

The `this` keyword is quite common in JavaScript and HTML Web pages, and unfortunately, it can be a source of confusion. The `this` keyword in JavaScript is determined as follows: If a method of an object `callingObject` caused the current block of code to be executed, then `this` is the object `callingObject`, unless the function reference is passed to an event handler, in which case `this` is the DOM element that declares the event handler. In an anonymous function `this` is the window object.

One of the really nice things about jQuery is that it simplifies the rather tricky rules regarding the `this` keyword.

---

**NOTE** *If you work with jQuery, you must use `$(this)` to invoke jQuery functions, and jQuery adds jQuery-specific functions to the `$(this)` object.*

The next part of this chapter shows you how to add mouse-related event handlers to create visual effects that respond to user gestures.

## D3 AND MOUSE EVENTS

This section shows you how to handle several mouse-related events in an HTML5 Web page with D3 code.

Listing 2.5 displays the contents of `ResizeCircle1.html` that illustrates how to resize an SVG `<circle>` element during mouse events.

### LISTING 2.5: *ResizeCircle1.html*

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8" />
    <title>Handling Mouse Events with D3</title>
    <script src="d3.min.js"></script>
  </head>

  <body>
    <script>
      d3.select("body")
        .append("svg")
        .attr("width", 300)
        .attr("height", 300)
        .append("circle")
        .attr("cx", 150)
        .attr("cy", 150)
        .attr("r", 30)
        .attr("fill", "blue")
        .on("mouseover", function() {
          return d3.select(this)
            .attr("r", 80)
            .attr("fill", "red");
        })
        .on("mouseout", function() {
          return d3.select(this)
            .attr("r", 50)
            .attr("fill", "green");
        })
        .on("click", function(d, i) {
          return console.log(d, i);
        });
    </script>
  </body>
</html>
```

Listing 2.5 starts with the usual boilerplate code followed by a `<script>` element that appends an SVG `<svg>` element and an SVG `<circle>` element to the `<body>` element of the current HTML Web page. Let's look at the D3 syntax to handle a `mouseover` event, as shown in the following code snippet:

```
.on("mouseover", function() {
  return d3.select(this)
    .attr("r", 80)
    .attr("fill", "red");
})
```

The key point to observe is that the `return` statement references the current element (which is an SVG `<circle>` element) with the `this` keyword and then changes the `r` attribute to 80 and the color to red. In a similar fashion, during the `mouseout` event, the value of `r` is set to 50 and the color changes to green. Finally, when users click on the circle, a message is displayed.

## MOUSE EVENTS AND RANDOMLY LOCATED TWO-DIMENSIONAL SHAPES

This section shows you how to create and render circles at random locations on the screen whenever users move their mouse on the screen. Listing 2.6 displays the contents of `RandomCircles1.html` that illustrate how to create and render SVG `<circle>` elements at random locations during mouse events.

### LISTING 2.6: *RandomCircles1.html*

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8" />
    <title>Rendering Random Circles with D3</title>
    <script src="d3.min.js"></script>
  </head>

  <body>
    <script>
      var width = 800, height = 500;
      var radius = 30, moveCount = 0, index = 0;
      var circleColors = ["red", "yellow", "green", "blue"];

      var svg = d3.select("body")
        .append("svg")
        .attr("width", width)
        .attr("height", height);

      svg.on("mousemove", function() {
        index = (++moveCount) % circleColors.length;

        var circle = svg.append("circle")
          .attr("cx", (width-100)*Math.random())
          .attr("cy", (height-100)*Math.random())
          .attr("r", radius);

        circle.attr("fill", circleColors[index]);
      });
    </script>
  </body>
</html>
```



Listing 2.6 bears some resemblance to Listing 2.5, with code to handle a `mousemove` event. The key idea is illustrated in the following code snippet:

```
var circle = svg.append("circle")
    .attr("cx", (width-100)*Math.random())
    .attr("cy", (height-100)*Math.random())
    .attr("r", radius);
```

The preceding code snippet uses the `Math.random()` function to generate random values that are assigned to the attributes `cx` and `cy` (the coordinates of the center point of a circle) as well as assigning the value of the variable `radius` to the attribute `r` (the radius of the circle). This code snippet is executed whenever users move their mouse (which can render many circles on the screen).

You can use slightly different syntax to reference a JavaScript function for handling an event. For example, compare the following code snippet with the `mousemove` code in Listing 2.6:

```
svg.on("mousemove", addCircle);

function addCircle() {
    // insert the circle-related code here
}
```

## A “FOLLOW THE MOUSE” EXAMPLE

This section shows you how to create and render circles at the current location on the screen whenever users move their mouse. Listing 2.7 displays the contents of `FollowTheMouse2.html` that illustrate how to create and render SVG `<circle>` elements at the current mouse location.

### **LISTING 2.7: *FollowTheMouse2.html***

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8" />
    <title>Follow the Mouse with D3</title>
    <script src="d3.min.js"></script>
  </head>

  <body>
    <script>
      var width = 800, height = 500;
      var stripWidth = 20, stripCount = 0;
      var radius = 30, moveCount = 0, index = 0;
      var factor = 0, factors = [1.0, 0.5];

      var circleColors = ["red", "yellow", "green", "blue"];

      var svg = d3.select("body")
        .append("svg")
        .attr("width", width)
        .attr("height", height);
```

```

svg.on("mousemove", function() {
    index = (++moveCount) % circleColors.length;

    stripCount = Math.floor(moveCount/stripWidth);
    factor = factors[stripCount%2];

    var circle = svg.append("circle")
        .attr("cx", d3.event.pageX)
        .attr("cy", d3.event.pageY)
        .attr("r", radius*factor);

    circle.attr("fill", circleColors[index]);
});
</script>
</body>

</html>

```

Listing 2.7 contains a `<script>` element with code that is familiar to you from previous code samples in this chapter. The key idea is to determine the current location of the user's mouse during a `mousemove` event and then append a new SVG `<circle>` at that location to the `<svg>` element of the current HTML Web page. This is accomplished by the following code snippet:

```

var circle = svg.append("circle")
    .attr("cx", d3.event.pageX)
    .attr("cy", d3.event.pageY)
    .attr("r", radius*factor);

```

As you can see in the preceding code snippet, the `D3.attr()` method uses the values `d3.event.pageX` and `d3.event.pageY` to set the values of the attributes `cx` and `cy` of a newly created SVG `<circle>` element, thereby creating a “follow the circle” effect.

While you can use the `pageX` and `pageY` that are available in the native event, transforming the event position to the local coordinate system of the container that received the event gives you some benefits. For instance, if you embed an SVG in the normal flow of an HTML Web page, you may want the event position relative to the top-left corner of the SVG image. In addition, if the SVG code contains a `transform` effect, you might want the position of the event relative to those transforms. Use the `d3.mouse` operator for the standard mouse pointer, and use `d3.touches` for multitouch events on iOS.

The preceding paragraph was taken (and slightly modified) from the following Website: <https://github.com/mbostock/d3/wiki/Selections#wiki-on>.

## A DRAG-AND-DROP EXAMPLE

If you have worked with drag-and-drop (DnD) in JavaScript, you will be happy to discover that D3 supports DnD functionality that is both simpler and

more robust than the HTML5 version. Moreover, D3 DnD functionality uses a syntax that is similar to other mouse-related events.

Before delving into a DnD code sample, it's worth noting that you can prevent the default DnD behavior (if you need to do so) with this code snippet:

```
function preventBehavior(e) {
  e.preventDefault();
}
document.addEventListener("touchmove", preventBehavior, false);
```

Listing 2.8 displays the contents of `MouseDownAndDrop1.html` that illustrate how to drag and drop a circle in an HTML Web page using D3.

### ***LISTING 2.8: MouseDragAndDrop1.html***

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8" />
  <title>Drag-and-Drop with D3</title>
  <script src="d3.min.js"></script>
</head>

<body>
  <script>
    var width=600, height=400, circle1;

    var drag = d3.behavior.drag()
      .on('dragstart', dragStart)
      .on('drag',      dragElement)
      .on('dragend',   dragEnd);

    d3.select("#circle1").call(drag);

    function dragStart(d, i) { }

    function dragElement(d, i) {
      circle1.attr("cx", d3.event.sourceEvent.pageX)
        .attr("cy", d3.event.sourceEvent.pageY);
    }

    function dragEnd(d, i) { }

    var svg = d3.select("body")
      .append("svg")
      .attr("width", width)
      .attr("height", height);

    circle1 = svg.append("circle")
      .attr("id", "circle1")
      .attr("cx", 50)
      .attr("cy", 50)
      .attr("r", 30)
```

```

        .attr("fill", "blue")
        .call(drag);

</script>
</body>
</html>

```

Listing 2.8 contains a `<script>` element that uses the `d3.behavior.drag()` method to define drag-related behavior that is attached to an SVG `<circle>` element, as shown here:

```

var drag = d3.behavior.drag()
    .on('dragstart', dragStart)
    .on('drag',      dragElement)
    .on('dragend',   dragEnd);

d3.select("#circle1").call(drag);

```

The preceding code snippet binds the JavaScript functions `dragStart()`, `dragElement()`, and `dragEnd()` to the mouse events `dragstart`, `drag`, and `dragend`, respectively. The JavaScript `dragElement()` function in Listing 2.8 is the only one that does something. This function updates the values of the attributes `cx` and `cy` with the current location of a user's mouse using the identical code that you saw in the preceding section:

```

circle1.attr("cx", d3.event.sourceEvent.pageX)
        .attr("cy", d3.event.sourceEvent.pageY);

```

Another example of handling drag events in D3 is here: <http://bl.ocks.org/mbostock/3680958>.

The D3 drag-handling code provides other functionality, and the source code is here: <https://github.com/mbostock/d3/blob/master/src/behavior/drag.js>.



The CD contains additional DnD code samples such as the HTML Web page `DragAndDropTheDots1.html` that combines mouse events, delayed animation, and scaling effects and applies them to all the circles in the code sample. The simple yet clever way to render the initial set of circles in a grid-like layout (using a combination of the `d3.range()` method and the `d3.map()` method) is from a code sample by Mike Bostock: <http://bl.ocks.org/mbostock/1557377>.

Because these preceding code samples use D3 animation effects, you can read their contents after reading the next section, which introduces you to animation effects in D3.

## ANIMATION EFFECTS WITH D3

The code sample in this section shows you how to create simple animation effects with D3. Listing 2.9 displays the contents of `Transition1.html` that illustrates how to render rectangles and programmatically change its position and dimensions to create an animation effect.

**LISTING 2.9: Transition1.html**

```

<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8" />
    <title>Animation Effects with D3</title>
    <script src="d3.min.js"></script>
  </head>

  <body>
    <script>
      var width    = 600, height = 400;
      var sWidth   = 100, sHeight = 100;
      var offsetX  = 100, offsetY = 100;
      var newWidth = 200, opacity = 0.5;
      var duration = 2000, delay  = 400; // milliseconds

      var svg = d3.select("body")
        .append("svg:svg")
        .attr("width", width)
        .attr("height", height);

      var mySquare = svg.append("rect")
        .attr("x", offsetX)
        .attr("y", offsetY)
        .attr("width", sWidth)
        .attr("height", sHeight)
        .style("fill", "red");

      mySquare
        .transition()
        .duration(duration)
        .style("opacity", opacity);

      svg.on("click", function() {
        mySquare
          .transition()
          .duration(duration)
          .delay(delay)
          .attr("x", 20)
          .attr("y", 20)
          .style("fill", "blue")
          .style("opacity", 1.0)
          .attr("width", newWidth);
      })
    </script>
  </body>
</html>

```

Listing 2.9 contains a `<script>` element that starts by defining an SVG `<rect>` element. Next, Listing 2.10 changes the value of the `opacity` attribute from 1 (the default value) to 0.5, as shown here:

```

mySquare
  .transition()

```

```
.duration(duration)
.style("opacity", opacity);
```

As you can see, the preceding code snippet uses the D3 `.transition()` function to change the `opacity` attribute because it is specified in the `.style()` method that immediately follows the `.transition()` method in the code snippet.

Finally, Listing 2.9 specifies a click event handler for the SVG `<rect>` element that changes the values of the attributes `x`, `y`, `fill`, `opacity`, and `width` to the values that are specified in the final code block in Listing 2.10.

Incidentally, a very good animation sequence is here, and after you have read the book chapters covering bar charts and graphs, it's worth your time to read the code that creates the multiple animation effects: <http://bl.ocks.org/mbostock/1256572>.

## EASING FUNCTIONS IN D3

In nontechnical terms, an easing function defines the manner in which an animation effect is created. In real life, the perception of motion depends on the vantage point of the viewer. For example, if you are in an airplane that is one thousand meters in the air, the cars below will appear to move at a constant rate. In D3, you can simulate this type of motion with the linear easing function because the transition from the start state to the end state is performed in constant time.

As a second (and more complex) example of perceived motion, imagine you are directly facing a baseball player (from a safe distance) and you watch the player hit a baseball into the air. The ball initially seems to move very quickly, then slow down and almost hang in the air for a short period of time, and then quickly drop to the ground.

There are several other concepts that you will encounter when you work with easing functions. The first is *ease-in*, which means start slow and speed up. The second is *ease-out*, which means start fast and slow to a stop. The third is *ease-in-out*, which is a combination of the previous two: start slow, speed up, and then slow down again.

Now that you have a basic understanding of easing functions and how they create different motion effects, you'll be relieved to discover that D3 supports an extensive set of easing functions including:

```
Linear: the identity function t.
poly(k): raises t to the specified power k (e.g., 3).
quad: equivalent to poly(2)
cubic: equivalent to poly(3)
sin: applies the trigonometric function sin
exp: raises 2 to a power based on t
```

You can specify an easing function in D3 as follows:

```
.ease("cubic"); // or any other easing function
```

Navigate to the following link to see a complete list of D3 easing functions and their description: [https://github.com/mbostock/d3/wiki/Transitions#wiki-d3\\_ease](https://github.com/mbostock/d3/wiki/Transitions#wiki-d3_ease).



The CD also contains a set of code samples that illustrate how to use the D3 easing functions with animation effects. The easing function is embedded in the file name. For example, the HTML Web page `MouseMoveFadeAnim1Bounce1.html` shows you how to use the bounce easing function, which simulates the motion of a bouncing object.

## ZOOM, PAN, AND RESCALE EFFECTS WITH D3

This section does not delve into the details of how to create zoom, pan, and rescale effects, but because this type of functionality might be useful for some people, some links are included so that you can read the D3 code.

Although the following code sample for creating a zoom effect is more sophisticated than the other examples in this chapter, you have seen almost every D3 method in the code (except for working with CSV files, which is covered in Chapter 4):

<http://mbostock.github.com/d3/talk/20111116/pack-hierarchy.html>.

The following HTML Web page illustrates how to create zoom, pan, and rescale effects: <https://gist.github.com/stephenb/1182434>.

Note that the preceding Web page contains a line graph, so the code will make more sense to you after reading about line graphs in chapter 4.

## HANDLING KEYBOARD EVENTS WITH D3

In addition to mouse events and touch events, D3 supports keyboard events, which involves examining the value of `d3.event.keyCode`. Listing 2.10 displays the contents of `Keyboard1.html` that illustrate how to detect keyboard events and also provides examples of determining specific keys.

### LISTING 2.10: *Keyboard1.html*

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8"/>
    <title>D3 and Keyboard Events</title>
    <script src="d3.min.js"></script>
  </head>

  <body>
    <script>
      d3.select("body").on("keydown", function() {
        switch (d3.event.keyCode) {
          case 8: console.log("delete"); break;
          case 32: console.log("space"); break;
          case 33: console.log("page up"); break;
          case 34: console.log("page down"); break;
          case 37: console.log("left arrow"); break;
          case 39: console.log("right arrow"); break;
```

```

        case 65: console.log("lower a");      break;
        case 97: console.log("upper A");      break;
        default: return;
    }

    d3.event.preventDefault();
  });
</script>
</body>
</html>

```

Listing 2.10 is straightforward: a `switch` statement checks several well-known values that represent various keys on a keyboard and prints a message that is visible in the Web Inspector.

## ADDITIONAL CODE SAMPLES ON THE CD



The three HTML Web pages `Cone2Anim1.html`, `Pyramid2Anim1.html`, and `BezierCurves2Anim1.html` add animation effects to the HTML Web pages

`Cone2.html`, `Pyramid2.html`, and `BezierCurves2.html` that you saw in Chapter 1. The HTML Web page `SimpleSketch1.html` is a variation of the code in `FollowTheMouse1.html` that shows you how you can render freestyle sketches.

The HTML Web page `IterateArrays3.html` illustrates how to handle multiple mouse events and how to render a set of line segments to create a so-called Moiret effect.

The HTML Web page `D3Timer1.html` uses the `d3.timer()` method to create an animation effect that changes the opacity value for a set of randomly generated circles. You can easily modify this code sample to create fade-in and fade-out effects.

The HTML Web page `MouseMoveFadeAnimation1.html` detects a `mouseover` event, an easing function, and a judicious choice of opacity values to create a very pleasing animation effect.

The HTML Web page `MultiPartialBlueSphereCB5Anim.html` creates scaling-based animation effects and scales the set of partial spheres during `mouseover` events.

## SUMMARY

This chapter showed you how to work with multidimensional JavaScript arrays followed by an example of using such arrays in a scatter plot with D3. Next, you learned about scaling functions in D3 and the D3 Path Data Generator. You also learned how to handle mouse events followed by some mouse-related code samples (such as “follow the mouse”) with D3. Finally, you learned how to create animation effects in HTML Web pages using D3. The next chapter is devoted to rendering bar charts with D3.





## WORKING WITH BAR CHARTS IN D3

This chapter shows you how to use D3 to create various bar charts along with code samples that have mouse-related effects, filter effects, and animation effects. The first part of this chapter shows you how to create a basic horizontal bar chart followed by two vertical bar charts (the second one uses data scaling). The second part of this chapter shows you how to create charts with dynamic data and charts that respond to mouse-related events. The third part of this chapter contains code samples that show you how to render bar charts with 3D effects and animation effects. These 3D-based bar charts (and those that are on the CD) provide a foundation that you can extend to create full-featured 3D bar charts.



### A SIMPLE HORIZONTAL BAR CHART

---

Bar charts can be horizontal as well as vertical, and they can also provide interactive functionality. Because this is the first section in this chapter, the code sample shows you how to create a horizontal bar chart. Subsequent sections contain code samples with additional functionality.

Listing 3.1 displays the contents of `HBarChart1.html` that illustrate how to create a simple horizontal bar chart.

#### **LISTING 3.1: *HBarChart1.html***

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8" />
  <title></title>

  <script src="d3.min.js"></script>
```

```

<style>
  #myChart {
    left: 10px; top: 10px;
    position: absolute;
  }

  p { font-size: 32px; }
</style>
</head>

<body>
  <div id="myChart">
    <p>Simple Bar Chart in D3</p>
  </div>

  <script>
    var dataValues = [10,5,25,15,20,35];
    var multX = 10, color="white", padding="2px";
    var barColors = ["red", "green", "blue"];

    d3.select("#myChart")
      .append("div")
      .selectAll("div")
      .data(dataValues)
      .enter()
      .append("div")
      .style("width", function (d) {
        return d*multX + "px";
      })
      .style("background-color", function (d) {
        return barColors[d % barColors.length];
      })
      .style("border", "1px solid white")
      .style("color", color)
      .style("padding", "2px")
      .text(function (d) { return d; });
  </script>
</body>
</html>

```

Listing 3.1 starts with the usual boilerplate code followed by a `<script>` element that defines a JavaScript array `dataValues` of numbers and several other JavaScript variables, including an array of standard colors.

The main D3 code in Listing 3.1 consists of `TMCHID3` that appends a set of rectangles whose pixel widths are a multiple of the numbers in the `dataValues` array, as shown in the following code snippet:

```

.style("width", function (d) {
  return d*multX + "px";
})

```

The D3 `.style()` method sets the value of the `width` property of each rectangle by means of an anonymous function that returns the string `d*multX+"px"` for each rectangle. Because the JavaScript variable `multX`



```

    .yaxis path, .yaxis line {
      fill: none;
      stroke: blue;
      stroke-width: 2px;
      shape-rendering: crispEdges;
    }
  </style>
</head>

<body>
  <script>
    var width=600, height=300;
    var indentX=40, indentY=20;
    var xTickCount=10, yTickCount=10;

    var xValues = [75, 250, 350, 120, 600, 450, 125, 850];
    var yValues = [25, 150, 250, 320, 400, 550, 325, 815];

    // Step 1: create an SVG element
    var svg = d3.select("body")
      .append("svg")
      .attr("width", width)
      .attr("height", height);

    // Step 2: create scale functions
    var xScale = d3.scale.linear()
      .domain([0, d3.max(xValues)])
      .range([indentX, width-indentX]);

    var yScale = d3.scale.linear()
      .domain([0, d3.max(yValues)])
      .range([height-indentY, indentY]);

    // Step 3: define the x-axis and y-axis
    var xAxis = d3.svg.axis()
      .scale(xScale)
      .orient("bottom")
      .ticks(xTickCount);

    var yAxis = d3.svg.axis()
      .scale(yScale)
      .orient("left")
      .ticks(yTickCount);

    // Step 4: render the x-axis and y-axis
    // You *must* use CSS to display thin axes!
    svg.append("g")
      .attr("class", "xaxis")
      .attr("font-size", "12px")
      .attr("transform",
        "translate(0," + (height-indentY)+")")
      .call(xAxis);

    svg.append("g")
      .attr("class", "yaxis")
      .attr("font-size", "16px")

```

```

        .attr("transform",
              "translate("+indentX+",0)")
        .call(yAxis);
    </script>
</body>
<html>

```

Listing 3.2 contains some boilerplate code and a `<style>` element for styling the vertical and horizontal axis. The `<script>` element defines some JavaScript variables and arrays and then appends an `<svg>` element to the `<body>` element, all of which you have seen in earlier chapters.

The next portion of code defines the `xAxis` variable using the previously defined `xScale` and the JavaScript variable `xTickCount` (which specifies the number of tick marks on the horizontal axis), as shown here:

```

var xAxis = d3.svg.axis()
    .scale(xScale)
    .orient("bottom")
    .ticks(xTickCount);

```

The corresponding code for the vertical axis is very similar to the code for the horizontal axis, so it won't be discussed here.

Now that the vertical and horizontal axes have been defined, the next portion of code renders the horizontal axis, as shown here:

```

svg.append("g")
    .attr("class", "xaxis")
    .attr("font-size", "12px")
    .attr("transform",
          "translate(0," + (height-indentY) + ")")
    .call(xAxis);

```

Recall that the `translate` function takes two values: the first specifies the horizontal direction (which is 0 in the preceding code block), and the second specifies the vertical direction (which is an expression in the preceding code block). Thus, the horizontal axis is rendered near the bottom of the screen. Another point to observe is the use of the CSS (Cascading Style Sheets) `axis` selector that is defined in the `<style>` element. The best way to understand how this works is to remove this code snippet and observe the corresponding change in the rendered axes.

The code for rendering the vertical axis is similar, except the axis is rendered vertically and also shifted toward the right by `indentX` units.

By the way, if you want to reverse the direction of the labels for the vertical axis, replace this code:

```

var yScale = d3.scale.linear()
    .domain([0, d3.max(yValues)])
    .range([height-indentY, indentY]);

```

with this code:

```
var yScale = d3.scale.linear()
    .domain([0, d3.max(yValues)])
    .range([indent, height-indentY]);
```

You could make a similar change for the horizontal axis by changing the `xScale` definition in an analogous manner (but it would be unusual to do so).

**NOTE** *If you do not specify a CSS class (such as the one shown in Listing 3.2) for an axis, it will be rendered with a thick black line.*

If you need more examples of labeling axes in D3, an extensive set of D3-based axes (including code) can be found here: <http://bl.ocks.org/GerHobbelt/3605124>.

Now that you understand how to render labeled horizontal and vertical axes, let's look at an example of rendering a scaled bar chart with labeled axes, which is discussed in the next section.

## A SCALED VERTICAL BAR CHART WITH LABELED AXES

Let's look at an example of rendering a simple vertical bar chart. One point to keep in mind is that while it's possible to create a horizontal bar chart with pure CSS3, it's more difficult to create a vertical bar chart, especially when you add various bells and whistles whose implementation would require very complex manipulations in CSS3.

Whenever you render a bar chart, you invariably need to scale the values of the heights so the chart will render correctly on a viewing device. As a simple example, if the heights of the bar elements of a chart are between 20 and 2,000 (or some other large range), then you obviously need to scale those values so your chart will be fully visible in the viewing area of your device.

The previous example showed you how to scale a set of numbers in a JavaScript array. In this section, the only substantive differences are how to generate the numbers in the `xValues` JavaScript array and the section of code for rendering the bar elements.

Listing 3.3 shows only the portion of `VBarChartScaledWithAxes1.html` that has been added to the code in Listing 3.2 (the entire code listing is available on the CD).



### LISTING 3.3: Code for Rendering a Scaled Bar Chart

```
<!DOCTYPE html>
<html>

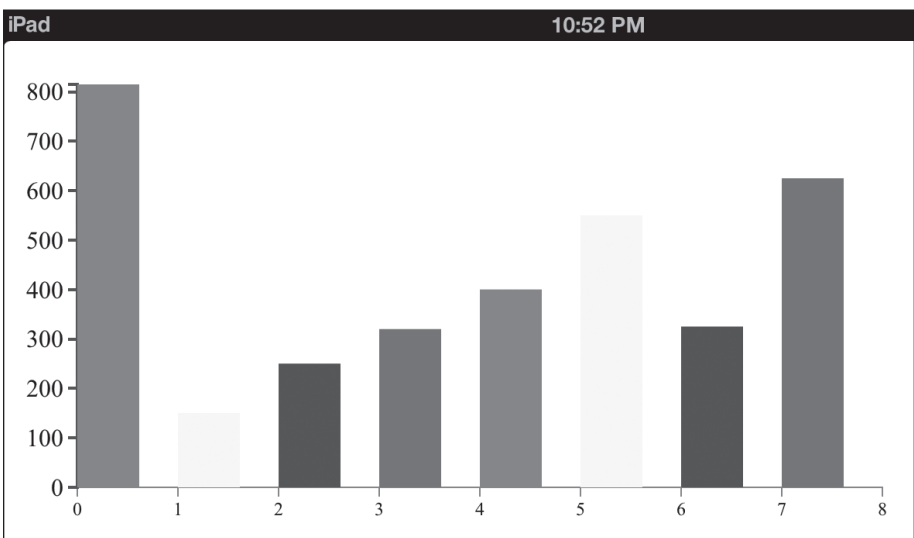
<body>
  <script>
    var yValues = [815, 150, 250, 320, 400, 550, 325, 625];
    var xValues = d3.range(yValues.length+1);
    var barColors = ["red", "yellow", "blue", "green"];
```

```
// Steps 1 through 4 are the same as Listing 3.1

// Step 5: render the bar chart elements
svg.selectAll("rect")
  .data(yValues)
  .enter()
  .append("rect")
  .attr("x", function(d, i) {
    return xScale(i);
  })
  .attr("y", function(y, i) {
    return yScale(y);
  })
  .attr("width", xScale(0))
  .attr("height", function(y, i) {
    return (yScale(0)-yScale(y));
  })
  .attr("fill", function(d, i) {
    return barColors[i%barColors.length];
  })
</script>
</body>
<html>
```

Listing 3.3 uses `TMCIID3` to render a set of rectangles by specifying the values for the attributes `x`, `y`, `width`, and `height`, all of which use the scaled values in `xScale` and `yScale` instead of the nonscaled chart values in `xValues` and `yValues`.

Figure 3.2 displays a vertical scaled bar chart with labeled axes on an iPad3.



**FIGURE 3.2** A Vertical Scaled Bar Chart with Labeled Axes on an iPad3.



## USING DATE AND TIME STAMPS TO LABEL AXES

The horizontal axis in Listing 3.2 uses the simplest possible set of numbers as labels. However, D3 supports a rich set of date and time formats that you can use as labels for vertical and horizontal axes.

As a simple example of working with date values, the following code block shows you three ways to convert the integer 2013 to a date expression:

```
var parseDate = d3.time.format("%Y").parse;
var year = 2013;

var aDate1 = parseDate(String(year));
var aDate2 = parseDate(year.toString());
var aDate3 = new Date(year, 0);
console.log("aDate1: "+aDate1);
console.log("aDate2: "+aDate2);
console.log("aDate3: "+aDate3);
```

In case you are wondering, the 0 in the definition for `aDate3` is required for the month of January. If you open the Web Inspector (see comments in the Preface) you will see the following output:

```
aDate1: Tue Jan 01 2013 00:00:00 GMT-0800 (PST)
aDate2: Tue Jan 01 2013 00:00:00 GMT-0800 (PST)
aDate3: Tue Jan 01 2013 00:00:00 GMT-0800 (PST)
```



The CD contains the HTML Web page `BarChartAndAxes2.html` that uses a set of dates to label the horizontal axis. There are two new sections of code, the first of which involves defining a set of dates in a JavaScript array:

```
var now = d3.time.hour.utc(new Date);

var timeValues = [
    d3.time.hour.utc.offset(now, -8),
    d3.time.hour.utc.offset(now, -7),
    d3.time.hour.utc.offset(now, -6),
    d3.time.hour.utc.offset(now, -5),
    d3.time.hour.utc.offset(now, -4),
    d3.time.hour.utc.offset(now, -3),
    d3.time.hour.utc.offset(now, -2),
    d3.time.hour.utc.offset(now, -1),
    d3.time.hour.utc.offset(now, 0)
];
```

As an example of what you can do with the preceding array, add the following snippet of code (which is used in the second code block below) and examine the output in the Web Inspector:

```
var aDate = d3.time.day.offset(
```

```
        new Date(timeValues[timeValues.length-1]), 1);
console.log("aDate: "+aDate);
```

The second new section of code involves defining the JavaScript variable `tScale` that is a time-based scale for the horizontal axis. The variable `tScale` is referenced in the definition of `xAxis`, as shown here:

```
// Step 3: define the x-axis and y-axis
var tScale = d3.time.scale()
    .domain([new Date(timeValues[0]),
        d3.time.day.offset(new Date(timeValues[timeValues.length-1]),1)]
    .rangeRound([1*indentX, width - 2*indentX]);

var xAxis = d3.svg.axis()
    .scale(tScale)
    .orient("bottom")
    .ticks(xTickCount)
    .tickFormat(d3.time.format("%m-%d"))
    // .tickFormat(d3.time.format("%Y-%m-%d"))
```

Try the commented-out code snippet that specifies a year/month/day format to see what happens to the labels on the horizontal axis.

Although the preceding code block uses a dash (-) between date-related values, you can use other characters (such as a slash [/] character), as well as the size of the tick mark, as shown here:

```
.tickSize(-height,0,0)
.tickFormat(d3.time.format("%m/%d %H:%M"));
```

As a fun exercise, try to guess what the following code snippet does and then include the code in an HTML Web page:

```
var xAxis = d3.svg.axis().scale(xRange)
    .tickSize(16).tickSubdivide(true),
var yAxis = d3.svg.axis().scale(yRange)
    .tickSize(10).orient("right").tickSubdivide(true);
```

Additional information about time formats and time scales can be found at <https://github.com/mbostock/d3/wiki/Time-Formatting> and <https://github.com/mbostock/d3/wiki/Time-Scales>.

## D3 BAR CHARTS WITH UNICODE CHARACTERS



D3 provides Unicode support for characters in languages such as Chinese and Japanese (and many others). The HTML Web page `BarChartAndAxes3 Unicode.html` on the CD shows you how to do the following: render Chinese and Japanese text using a Unicode escape sequence, add labels to horizontal and vertical axes, and display “jumbled” text.

If you are unfamiliar with using Unicode to represent characters of other languages, an example of representing Chinese and Japanese characters in Unicode (which is also used in the code sample on the CD) is as follows:



```
var chinese1 = "\u5c07\u63a2\u8a0e HTML5 \u53ca\u5176\u4ed6";
var hiragana =
    "D3 \u306F \u304B\u3063\u3053\u3043\u3043 \u3067\u3059!";
```

The D3 code for rendering the preceding Japanese text string is the same code that you would use to render any other character string, as shown here:

```
svg.append("g")
    .append("text")
    .attr("id", "japanese")
    .attr("text-anchor", "middle")
    .text(hiragana)
    .attr("font-size", "24px")
    .attr("text-anchor", "middle")
    .attr("x", width/2)
    .attr("y", height/2)
```

Figure 3.3 displays the output from `BarChartAndAxes3Unicode.html` in a Chrome browser on a Macbook Pro.

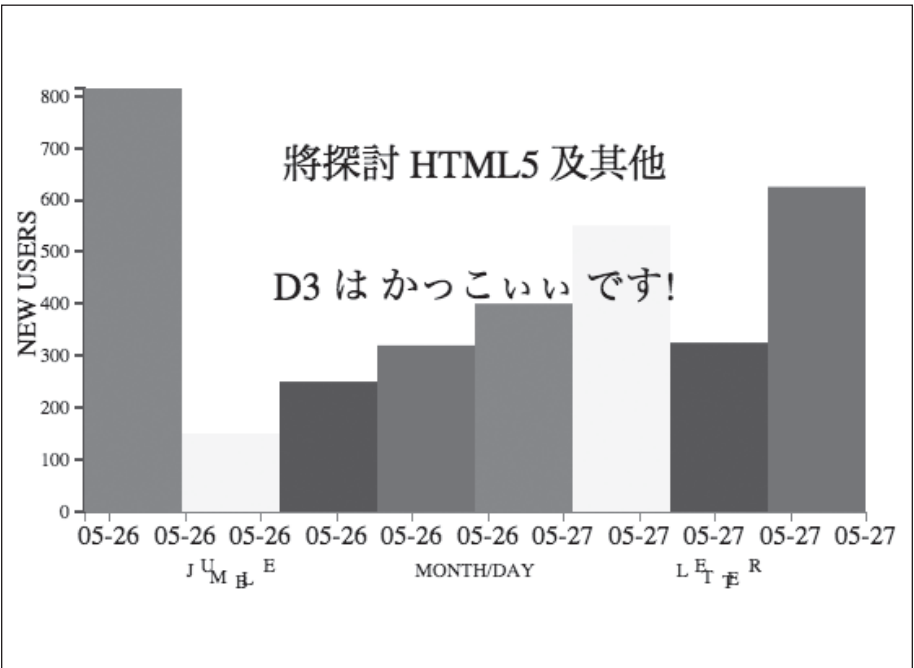


FIGURE 3.3 Unicode and Titles for Axes in D3.

## BAR CHARTS WITH THREE-DIMENSIONAL EFFECTS

A 3D bar chart is a set of contiguous 3D bar elements where each bar element is represented by three faces: a front face (a rectangle), a top face (a parallelogram), and a right face (also a parallelogram). In Chapter 1, you saw how to render a polygon in D3 by creating an SVG `<path>` element (see `Polygon1.html` for details). Because a parallelogram is also a polygon, you can use an SVG `<path>` element to render the top face and the right face of each 3D bar element and an SVG `<rect>` element to render the front face.

Listing 3.4 displays the contents of `Multi3DBarChart1.html` that illustrate how to create and render a bar chart with a 3D effect. You might be surprised to discover that this HTML Web page contains fewer than one hundred lines of code.

### **LISTING 3.4: *Multi3DBarChart1.html***

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <title>A 3D Multi-Bar Chart</title>
  <script src="d3.min.js"></script>
</head>

<body>
  <script>
    var width=800, height=500;
    var basePointX=200, basePointY=0;
    var currentX=0, currentY=0, s=0;
    var minHeight=20, maxHeight=160;
    var barCount=20, barWidth=12;
    var slantX=barWidth/3, slantY=barWidth/3;
    var rowCount=8, colCount=6, shiftX=0, shiftY=0;
    var points1="", points2="";
    var otherDirectionRow=0, interRowGapFactor=4;
    var barHeights = new Array(barCount);
    var multiBarHeights = new Array(rowCount);
    var topColors = ["#eeeeee", "#ffcccc", "#ccccff"];
    var barColors = ["#ff4444", "#d4bb44", "#4444ff"];
    var sideColors = ["#4444cc", "#4444cc"];

    var svg = d3.select("body")
      .append("svg")
      .attr("width", width)
      .attr("height", height);

    var g1 = d3.select("svg").append("svg:g");
    g1.attr("id", "g1")
      .attr("transform", "scale(0.5,0.5)");

    for(var row=0; row<rowCount; row++) {
      multiBarHeights[row] = new Array(barCount);
```

```

    for(var b=0; b<barCount; b++) {
        barHeights[b] =
            Math.floor((maxHeight-minHeight)*Math.random()+
                minHeight);

        multiBarHeights[row][b] = barHeights[b];
    }
}

for(var row=rowCount-1; row>=0; row--) {
    barHeights = multiBarHeights[row];

    for(var b=0; b<barCount; b++) {
        shiftX = interRowGapFactor*otherDirectionRow*slantX;
        shiftY = interRowGapFactor*otherDirectionRow*slantY;
        offsetX = basePointX+b*barWidth-shiftX;
        offsetY = basePointY+(maxHeight-barHeights[b])+shiftY;

        // CCW from lower left vertex
        points1 = offsetX+","+offsetY+" "+
            (offsetX+barWidth)+","+offsetY+" "+
            (offsetX+barWidth+slantX)+","+ (offsetY-slantY)+" "+
            (offsetX+0*barWidth+slantX)+","+ (offsetY-slantY);

        // CW from lower left vertex
        points2 = (offsetX+barWidth)+","+offsetY+" "+
            (offsetX+barWidth+slantX)+","+ (offsetY-slantY)+" "+
            (offsetX+barWidth+slantX)+","+ "+
            (offsetY-slantY+barHeights[b])+"+ "+
            (offsetX+barWidth)+","+ (offsetY+barHeights[b]);

        var gc = gl.append("svg:g");

        // top face (parallelogram)
        var polygon = gc.append("polygon")
            .attr("points", points1)
            .attr("fill",
                topColors[(s+b+1)%topColors.length])

        // front face (rectangle)
        gc.append("svg:rect")
            .attr("x", offsetX)
            .attr("y", offsetY)
            .attr("width", barWidth)
            .attr("height", barHeights[b])
            .attr("fill", barColors[(s+b)%barColors.length])

        // right face (parallelogram)
        gc.append("polygon")
            .attr("points", points2)
            .attr("fill",
                sideColors[(s+b+2)%sideColors.length])
    }
}

```

```

        gc.on("mouseover", function() {
            d3.select(this)
                .attr("transform", "scale(0.5,0.5)")
        })
    }
    ++otherDirectionRow;
}
</script>
</body>
</html>

```

Listing 3.4 consists of five steps that are identified as follows:

```

// STEP 0: INITIALIZATION
// STEP 1: CREATE THE "TOP FACE" OF THE BAR ELEMENTS
// STEP 2: CREATE THE "RIGHT FACE" OF THE BAR ELEMENTS
// STEP 3: DRAW THE "FRONT FACE" OF THE BAR ELEMENTS
// STEP 4: ADD LABELS TO THE BAR ELEMENTS

```

Listing 3.4 starts with boilerplate code followed by a `<script>` element that initializes a set of JavaScript variables. After creating an SVG `<svg>` element and an SVG `<g>` element, Listing 3.3 initializes the JavaScript variable `multiBarHeights` (which is a multidimensional array) with randomly generated values.

The next loop in Listing 3.4 consists of almost fifty lines of code to generate multiple rows of 3D bar elements, from back to front, based on the random values. The bar element in each row consists of a top face (a parallelogram), a front face (a rectangle), and a right face (also a parallelogram). The JavaScript `points1` variable is a string consisting of four points starting from the lower left vertex and proceeding in a counter-clockwise (CCW) fashion. This string represents the top face of the current 3D bar element, and it's constructed as shown here:

```

// CCW from lower left vertex
points1 = offsetX+","+offsetY+" "+
    (offsetX+barWidth)+","+offsetY+" "+
    (offsetX+barWidth+slantX)+","+ (offsetY-slantY)+" "+
    (offsetX+0*barWidth+slantX)+","+ (offsetY-slantY);

```

The string `points1` is used as the value of the attribute `points` to create the SVG `<polygon>` element in D3 for the top face, as shown here:

```

// top face (parallelogram)
var polygon = gc.append("polygon")
    .attr("points", points1)
    .attr("fill", topColors[(s+b+1)%topColors.length])

```

Similarly, the JavaScript `points2` variable is a string consisting of four points starting from the upper left vertex and proceeding in a clockwise (CW)

fashion. This string represents the right face of the current 3D bar element, and it's constructed as shown here:

```
// CW from lower left vertex
points2 = (offsetX+barWidth)+", "+offsetY+" "+
          (offsetX+barWidth+slantX)+", "+(offsetY-slantY)+" "+
          (offsetX+barWidth+slantX)+", "+
          (offsetY-slantY+barHeights[b])+", "+
          (offsetX+barWidth)+", "+(offsetY+barHeights[b]);
```

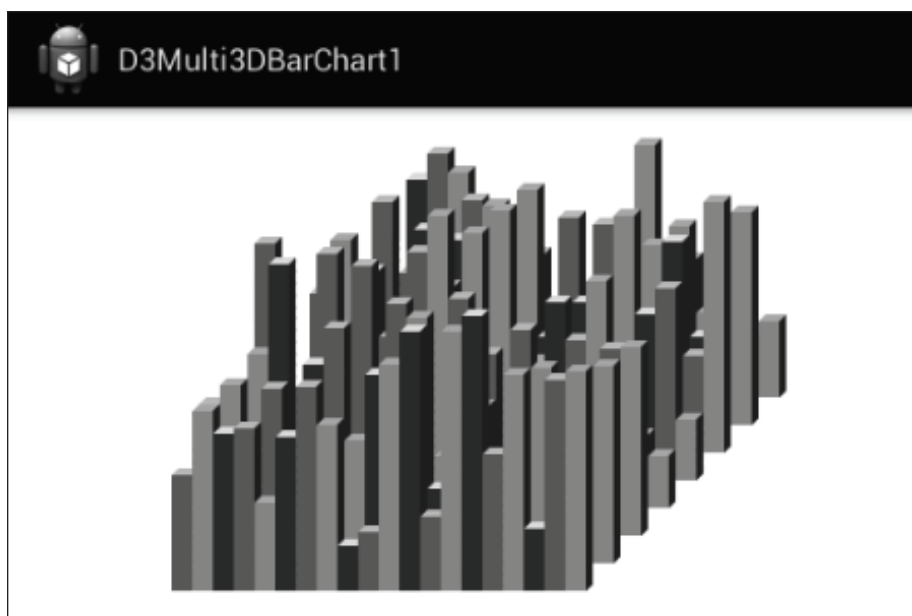
The JavaScript variable `points2` is used as the value of the attribute `points` in an SVG `<polygon>` element that is used in D3 to render the right face, as shown here:

```
// right face (parallelogram)
gc.append("polygon")
  .attr("points", points2)
  .attr("fill", sideColors[(s+b+2)%sideColors.length])
```

The final code snippet in this loop is an event handler for a `mouseover` event for each 3D bar element, which scales the bar element to half its original width and height, as shown here:

```
gc.on("mouseover", function() {
  d3.select(this)
    .attr("transform", "scale(0.5,0.5)")
})
```

Figure 3.4 displays a 3D multibar chart on an iPad3.



**FIGURE 3.4** A 3D Multibar Chart on an iPad3.



The CD contains additional 3D-related bar charts that use other D3 functionalities including animation effects.

## BAR CHARTS WITH FILTER EFFECTS AND TOOLTIPS

Now that you know how to render horizontal and vertical bar charts, you are ready to learn how to create better visual effects. One way to improve the appearance of a bar chart (and other types of graphs) is to create an SVG `<filter>` element. SVG has extensive support for filters, and the code sample in this section shows you how to use a Gaussian blur filter.

Listing 3.5 displays the contents of `VBarChart2Filter1.html` that illustrate how to create and render a bar chart with a filter effect. The code for defining a filter (and applying it to each bar element of the chart) is shown in bold.

### **LISTING 3.5: *VBarChart2Filter1.html***

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8" />
  <title></title>

  <script src="d3.min.js"></script>

  <style>
    svg {
      position: absolute;
      left: 40px; top: 10px;
    }
  </style>
</head>

<body>
  <script>
    var width = 480, height = 400, barPadding = 8;
    //var dataValues = [10, 150, 20, 280, 120, 90, 30, 60];
    var dataValues = [100, 250, 220, 380, 220, 190, 230, 360];
    var barColors = ["red", "yellow", "blue", "green"];
    var indentX = 5, indentY = 5;

    // create the <svg> element...
    var svg = d3.select("body")
      .append("svg")
      .attr("width", width)
      .attr("height", height);

    // scale the data values...
    var x = d3.scale.ordinal()
      .domain(dataValues)
      .rangeBands([0, width - (barPadding*dataValues.length)]);

    var y = d3.scale.linear()
      .domain([0,width])
      .range([height, 0])
```



```

var filter = svg.append("svg:defs")
    .append("svg:filter")
    .attr("id", "blurFilter1")
    .append("svg:feGaussianBlur")
    .attr("stdDeviation", 4);

// draw the chart elements...
svg.selectAll("rect")
    .data(dataValues)
    .enter()
    .append("rect")
    .attr("x", function(d, i) {
        return i * (width/dataValues.length);
    })
    .attr("y", function(y) {
        return y;
    })
    .attr("width", x.rangeBand())
    .attr("height", y)
    .attr("fill", function(d, i) {
        return barColors[i%barColors.length];
    })
    .attr("filter", "url(#blurFilter1)")
    .append("svg:title")
    .text(function() {
        var parent = d3.select(this).node().parentNode;
        var h2 = d3.select(parent).attr("height");
        return Math.floor(h2);
    })

// label the height of each bar element
svg.selectAll("circle")
    .data(dataValues)
    .enter()
    .append("text")
    .text(function(d) {
        return height-d;
    })
    .attr("text-anchor", "middle")
    .attr("x", function(d, i) {
        var offsetX = i*(width/dataValues.length);
        return offsetX+(width/dataValues.length)/2;
    })
    .attr("y", function(y) {
        return y;
    })

// draw the horizontal axis...
svg.append("line")
    .attr("x1", 0)
    .attr("x2", width * dataValues.length+indentX)
    .attr("y1", height - 1.0)
    .attr("y2", height - 1.0)
    .style("stroke-width", 4)
    .style("stroke", "#000");
</script>

```

```

</body>
</html>

```

Listing 3.5 renders a vertical bar chart using the same code as Listing 3.4, so that code will not be discussed again. The new code in Listing 3.5 involves defining a filter and specifying this filter in each bar element of the vertical bar chart. The definition of the filter is shown here:

```

var filter = svg.append("svg:defs")
                .append("svg:filter")
                .attr("id", "blurFilter1")
                .append("svg:feGaussianBlur")
                .attr("stdDeviation", 4);

```

Although the preceding code snippet might look complicated, the code corresponds to the following SVG `<defs>` element (which is probably easier to understand):

```

<defs>
  <filter id="blurFilter1">
    <feGaussianBlur stdDeviation="3"/>
  </filter>
</defs>

```

The next portion of code in Listing 3.5 uses `TMCIID3` to create a set of vertical bar elements for the bar chart, and the filter effect on each bar element is specified with this code snippet:

```

.attr("filter", "url(#blurFilter1)")

```

There is another new concept in Listing 3.5: the addition of a tooltip, which displays the height of the current bar element when users hover over it with their mouse. This effect is created with the following code block:

```

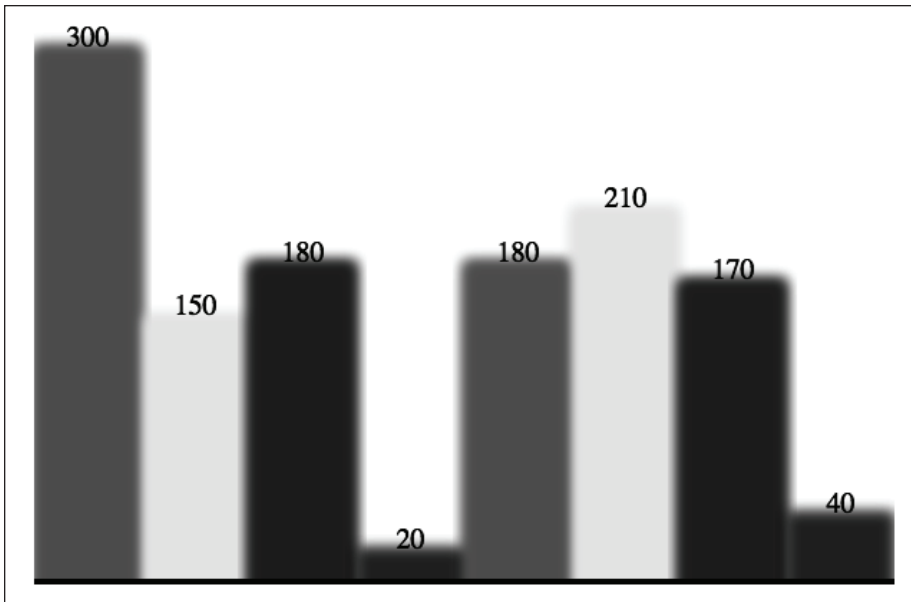
.append("svg:title")
  .text(function() {
    var parent = d3.select(this).node().parentNode;
    var h2 = d3.select(parent).attr("height");
    return Math.floor(h2);
  })

```

Figure 3.5 displays a vertical bar chart with a tooltip effect in a Chrome browser on a Macbook Pro.

## ADDITIONAL FILTER EFFECTS

SVG provides a rich set of filters to create visually appealing effects in Web pages. There are two points to keep in mind: First, there can be a performance impact (especially for those that create richer visual effects), so use



**FIGURE 3.5** A Vertical Bar Chart with a Filter in a Chrome Browser on a Macbook Pro.

them judiciously. You quickly can test your Websites by noting the difference in speed when you use various filters. Second, the Website <http://caniuse.com/svg-filters> provides a summary of the support (and the level of support) for SVG filters in desktop browsers and mobile browsers.



With the preceding points in mind, the CD contains an assortment of code samples with SVG filters. For example, the HTML Web page `Turbulence-Filter1.html` illustrates how to define and use a turbulence filter (which is much more complex than the filter in Listing 3.5) in D3. A code fragment with the filter definition in this code sample is shown here:

```
var filter = svg.append("svg:defs")
    .append("svg:filter")
    .attr("id", "turbFilter1")
    .attr("in", "SourceImage")
    .attr("filterUnits", "objectBoundingBox");

var turb1 = filter.append("svg:feTurbulence")
    .attr("baseFrequency", 0.05)
    .attr("numOctaves", 1)
    .attr("result", "turbulenceOut1");

var disp1 = filter.append("svg:feDisplacementMap")
    .attr("in", "SourceGraphic")
    .attr("in2", "turbulenceOut1")
    .attr("xChannelSelector", "B")
    .attr("yChannelSelector", "R")
    .attr("scale", 2);
```

```

var diff1 = filter.append("svg:feDiffuseLighting")
    .attr("in", "SourceGraphic")
    .attr("diffuseConstant", 2)
    .attr("surfaceScale", 20)
    .attr("style", "lighting-color:#FFFFCC")
    .append("svg:feSpotLight")
    .attr("x", 30)
    .attr("y", 10)
    .attr("z", 80)
    .attr("pointsAtX", 30)
    .attr("pointsAtY", 30)
    .attr("pointsAtZ", 0)
    .attr("specularExponent", 4);

```

You can create lighting effects with two light filter primitives (`feDiffuseLighting` and `feSpecularLighting`) and three light sources (`feDistantLight`, `fePointLight`, and `feSpotlight`). An example is shown here:

```

<filter id="LightingFilter1">
  <feDiffuseLighting>
    <feSpotLight x="0" y="0" z="50"
      pointsAtX="300" pointsAtY="300" pointsAtZ="0"
      limitingConeAngle="20" specularExponent="5"/>
  </feDiffuseLighting>
</filter>

```

Another example of a lighting filter is shown here:

```

<filter id="LightingFilter2">
  <feDiffuseLighting result="result1">
    <feSpotLight x="0" y="0" z="50"
      pointsAtX="300" pointsAtY="300" pointsAtZ="0"
      limitingConeAngle="20" specularExponent="5"/>
  </feDiffuseLighting>
  <feComposite operator="arithmetic" k1="1" k2="0" k3="0" k4="0"
    in="SourceGraphic" in2="result1"/>
</filter>

```

An example of a shadow filter is shown here:

```

<filter id="ShadowFilter">
  <feOffset dx="5" dy="5"/>
  <feGaussianBlur stdDeviation="3"/>
  <feColorMatrix type="matrix"
    values="0 0 0 0 0 .2, 0 0 0 0 1, 0 0 0 0 .75, 0 0 0 1 0"
    result="shadow"/>
  <feMerge>
    <feMergeNode in="shadow"/>
    <feMergeNode in="SourceGraphic"/>
  </feMerge>
</filter>

```

Two examples of morphology filters are shown here:

```
<filter id="MorphologyFilter1" >
  <feMorphology operator="erode" radius="7" />
</filter />

<filter id="MorphologyFilter2" >
  <feMorphology operator="erode" radius="7" />
  <feMorphology operator="dilate" radius="6" />
</filter />
```

Two examples of color-matrix filters are shown here:

```
<filter id="ColorMatrixFilter1" >
  <feColorMatrix type="saturate" values="2" />
</filter />

<filter id="ColorMatrixFilter2" >
  <feColorMatrix type="hueRotate" values="180" />
</filter />
```

An example of an feFlood filter:

```
<filter id="FeFloodFilter" filterUnits="userSpaceOnUse"
  x="0" y="0" width="400" height="400">
  <feFlood x="100" y="100" width="100" height="100"
    flood-color="green" flood-opacity="0.5"/>
</filter>
```

An example of combining an feImage filter and an feTile filter is shown here:

```
<filter id="ImageTileFilter" filterUnits="userSpaceOnUse"
  x="0" y="0" width="400" height="400">
  <feImage x="0" y="0" width="100" height="100"
    xlink:href="sample.svg" result="picture"/>
  <feTile x="0" y="0" width="400" height="400" in="picture"/>
</filter>
```

The following example illustrates how to create sophisticated filter effects using a combination of SVG filters:

```
<filter id="MyFilter" filterUnits="userSpaceOnUse">
  <feGaussianBlur in="SourceAlpha" stdDeviation="4" result="blur"/>
  <feOffset in="blur" dx="4" dy="4" result="offsetBlur"/>
  <feSpecularLighting in="blur" surfaceScale="5"
    specularConstant=".75"
    specularExponent="20"
    lighting-color="#bbbbbb"
    result="specOut">
    <fePointLight x="-5000" y="-10000" z="20000"/>
  </feSpecularLighting>
```

```

<feComposite in="specOut" in2="SourceAlpha"
  operator="in" result="specOut"/>
<feComposite in="SourceGraphic" in2="specOut"
  operator="arithmetic"
  k1="0" k2="1" k3="1" k4="0" result="litPaint"/>
<feMerge>
  <feMergeNode in="offsetBlur"/>
  <feMergeNode in="litPaint"/>
</feMerge>
</filter>

```

You can use the filter in Listing 3.5 as a guide for converting any of the preceding SVG filters to their D3 counterparts.

Keep in mind that some SVG-based filters are very CPU intensive, so it's best to use filters judiciously in your applications. If you plan to develop hybrid mobile applications, keep in mind that support for SVG filters depends on the mobile device (and iOS generally has better support for filters than Android).

Chapter 5 provides an overview of SVG, and the lone filter-based example uses the same filter as Listing 3.5. In addition, an extensive collection of code samples with SVG-based filter effects can be found at <http://code.google.com/p/svg-filters-graphics/>. An open-source project with D3-based code samples can be found at <http://code.google.com/p/d3-graphics/>.

## UPDATING BAR CHARTS

---

The code samples in this section show you how to render a bar chart that refreshes the data whenever users click on the bar chart with their mouse. The data values in this code sample are randomly generated numbers that simulate live data values. In a production Website, you would replace the random values with actual values from an external source (such as a Web service). You will learn how to use external data sources in the code samples in Chapter 9.

Listing 3.6 displays the contents of `BarChart2RandomUpdates1.html` that illustrate how to refresh a bar chart with a new set of data values.

### **LISTING 3.6: *BarChart2RandomUpdates.html***

```

<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>Updating a Bar Chart with Random Values</title>
    <script src="d3.min.js"></script>
  </head>

  <body>
    <p>Click to update the bar chart:</p>

    <script>
      var width=600, height=300;

```

```

var offsetY=15, roundBand=0.05, theColor="";
var clickCount=0, weight=0, multiplier=100;
var textColor="white";

// initial values for the bar chart...
var dataValues = [20, 15, 30, 25, 12, 18, 28, 38,
                  21, 32, 26, 20, 18, 8, 16, 18];

var dataCount = dataValues.length;

var xScale = d3.scale.ordinal()
  .domain(d3.range(dataValues.length))
  .rangeRoundBands([0, width], roundBand);

var yScale = d3.scale.linear()
  .domain([0, d3.max(dataValues)])
  .range([0, height]);

//Create SVG element
var svg = d3.select("body")
  .append("svg")
  .attr("width", width)
  .attr("height", height);

svg.selectAll("rect")
  .data(dataValues)
  .enter()
  .append("rect")
  .attr("x", function(d, i) {
    return xScale(i);
  })
  .attr("y", function(d) {
    return height - yScale(d);
  })
  .attr("width", xScale.rangeBand())
  .attr("height", function(d) {
    return yScale(d);
  })
  .attr("fill", function(d) {
    return "rgb(0, 0, " + (d * 10) + ")";
  });

// Create bar labels...
svg.selectAll("text")
  .data(dataValues)
  .enter()
  .append("text")
  .text(function(d) {
    return d;
  })
  .attr("text-anchor", "middle")
  .attr("x", function(d, i) {
    return xScale(i) + xScale.rangeBand() / 2;
  })
  .attr("y", function(d) {
    return height - yScale(d) + offsetY;
  });

```

```

    })
    .attr("font-family", "sans-serif")
    .attr("font-size", "12px")
    .attr("fill", textColor);

// update bar chart with random values...
svg.on("click", function() {
    ++clickCount;

    // generate a set of random values...
    dataValues = [];
    for(var x=0; x<dataCount; x++) {
        randValue = multiplier*Math.random();
        dataValues.push(randValue);
    }

    // reset the scaled x and y values...
    xScale = d3.scale.ordinal()
        .domain(d3.range(dataValues.length))
        .rangeRoundBands([0, width], roundBand);

    yScale = d3.scale.linear()
        .domain([0, d3.max(dataValues)])
        .range([0, height]);

    // Update the rectangles...
    svg.selectAll("rect")
        .data(dataValues)
        .attr("y", function(d) {
            return height - yScale(d);
        })
        .attr("height", function(d) {
            return yScale(d);
        })
        .attr("fill", function(d) {
            weight = Math.floor((d*10) % 255);
            colorR = "rgb("+weight+", 0, 0)";
            colorG = "rgb(0,"+weight+",0)";
            colorB = "rgb(0, 0,"+weight+")";

            if(clickCount % 3 == 0) {
                theColor = colorR;
            } else if(clickCount % 3 == 1) {
                theColor = colorG;
            } else if(clickCount % 3 == 2) {
                theColor = colorB;
            }

            return theColor;
        });

    // Update the labels...
    svg.selectAll("text")
        .data(dataValues)
        .text(function(d) {

```



```

        //return d;
        return Math.floor(d);
    })
    .attr("x", function(d, i) {
        return xScale(i) + xScale.rangeBand()/2;
    })
    .attr("y", function(d) {
        return height - yScale(d) + offsetY;
    });
});
</script>
</body>
</html>

```

Listing 3.6 renders an initial vertical bar chart using the same code as Listing 3.5, so a discussion of that code will not be repeated here. The new code in Listing 3.6 involves replacing the current vertical bar chart with a new vertical bar chart whenever users click on the bar chart with their mouse.

The first portion of the D3 code that handles a mouse-click event is shown here:

```

svg.on("click", function() {
    ++clickCount;

    // generate a set of random values...
    dataValues = [];
    for(var x=0; x<dataCount; x++) {
        randValue = multiplier*Math.random();
        dataValues.push(randValue);
    }
}

```

The preceding code block sets the JavaScript array `dataValues` to an empty array followed by a simple loop that uses random values to populate the array with a new set of numbers.

The next portion of D3 code (which is also in the same click-handler function) resets the scale-related variables for the horizontal and vertical values of the bar elements as shown here:

```

// reset the scaled x and y values...
xScale = d3.scale.ordinal()
    .domain(d3.range(dataValues.length))
    .rangeRoundBands([0, width], roundBand);

yScale = d3.scale.linear()
    .domain([0, d3.max(dataValues)])
    .range([0, height]);

```

The preceding code is the same as the code you have seen in previous code samples, and it is required every time a new vertical bar chart is rendered.

The next portion of code in the click handler uses `TMCIID3` to render a set of bar elements based on the newly generated set of numbers in the

`dataValues` array. The color for each bar element is determined with the following code block:

```
.attr("fill", function(d) {
    weight = Math.floor((d*10) % 255);
    colorR = "rgb("+weight+", 0, 0)";
    colorG = "rgb(0, "+weight+", 0)";
    colorB = "rgb(0, 0, "+weight+")";

    if(clickCount % 3 == 0) {
        theColor = colorR;
    } else if(clickCount % 3 == 1) {
        theColor = colorG;
    } else if(clickCount % 3 == 2) {
        theColor = colorB;
    }

    return theColor;
});
```

The preceding code block uses the quantity `d*10 % 255` (which is an integer between 0 and 255) to compute an (R,G,B) color value. Notice that the variables `colorR`, `colorG`, and `colorB` contain a single nonzero component, which will render a variation of red, green, and blue, respectively.

The second portion of the preceding code block uses conditional logic to determine which color to assign to the variable `theColor` that is used to render the color of the current bar element.

Keep in mind that although the preceding block of code for setting the color is probably more complex than your needs, it's included here so you will learn how to incorporate this type of functionality in HTML Web pages if you need to do so.

The final block of code in the click-event handler sets the text labels to display the height of each bar element in the vertical bar chart.

Figure 3.6 displays an updated bar chart in a Chrome browser on a MacBook Pro.

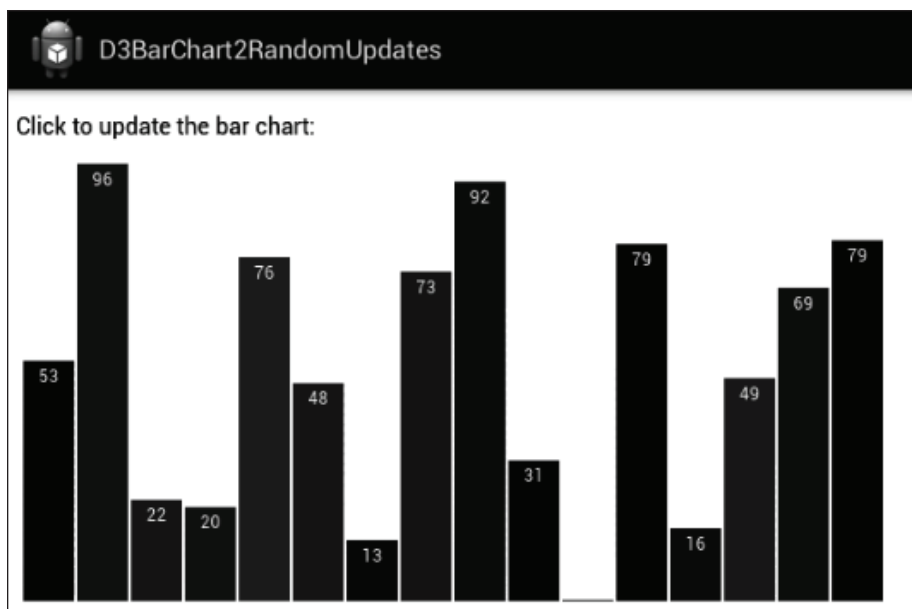
## DYNAMICALLY ADDING AND REMOVING DATA FROM BAR CHARTS

In most cases, you treat charts and graphs as read-only data that is retrieved from another source, so there are few cases where you need to add or remove data values from a chart or graph. However, if you do need this type of functionality, it's straightforward to implement in D3.

Listing 3.7 displays the logic that is required to add a bar element and to remove a bar element from a bar chart.

---

**NOTE** *Listing 3.7 does not have a standalone Web page on the CD (add the code to Listing 3.6 to create the desired effect).*



**FIGURE 3.6** An Updated Bar Chart in a Chrome Browser on an Android tablet.

### ***LISTING 3.7: Deleting and Adding Data from a Chart or Graph***

```
// delete the first data value:
// dataValues.shift();

// append a new data value:
dataValues.push(Math.floor(Math.random()*100));

// you need to reset the x scale:
xScale.domain(d3.range(dataset.length));

// you also need to reset the y scale:
yScale.domain([0, d3.max(dataset)]);
```

The code block in Listing 3.7 handles the case where you want to delete the first data value or append a new data value to an existing set of data points. If you need to remove or add data somewhere between the first and last data values, you can add a mouse-click event handler that determines which chart or graph element the users have clicked and then use code that is similar to Listing 3.6 to update the visual display accordingly.

In addition, the previous code samples provide you with sufficiently similar D3-based code that is required to implement this functionality.

## **SCROLLING ANIMATION EFFECTS WITH BAR CHARTS**

D3 enables you to create scrolling effects with bar charts, which can be very handy if you are working on time-sensitive live data, such as ticker-tape stock-related data or weather information.

Mike Bostock has written a very good introduction to scrolling techniques here, which can be found at <http://mbostock.github.com/d3/tutorial/bar-2.html>. You can use Mike's techniques to create similar effects for rendering other types of data visualization (such as line charts and stacked bar charts) that involve real-time data. You can also incorporate these techniques in the 3D-bar-chart code samples in Chapter 3 to create vivid graphics effects.

## ADDITIONAL CODE SAMPLES ON THE CD



The CD contains various code samples that show you how to create visual effects in D3 and SVG. The code samples that render polygons, cubes, and pyramids with blur filters and SVG `<pattern>` elements are included because they provide techniques that might be helpful for creating more vivid effects in your custom code. For example, the three polygon-based HTML Web pages `Polygon1Anim1.html`, `Polygon1Pattern1Anim1.html`, and `Polygon1BlurFilter1Anim1.html` add animation effects (sometimes mouse related) to the corresponding HTML Web pages without `Anim1` in the filenames.

The HTML Web page `TurbulenceFilter1.html` illustrates how to define a turbulence filter. The HTML Web pages `CubeBlurFilter1.html` and `PyramidBlurFilter1.html` render a cube and a pyramid, respectively, with a Gaussian blur filter. The HTML Web pages `Cube1Pattern1.html` and `Pyramid1Pattern1.html` render a cube and a pyramid, respectively, with an SVG `<pattern>` element as well as a Gaussian blur filter.

The HTML Web page `Multi3DTBarChart1.html` renders a 3D bar chart where each bar element is a triangular pyramid instead of a parallelepiped (often called a box) for the bar elements. The HTML Web pages `Multi3DBarChart1Filter1.html` and `Multi3DBarChart1LRGrad1.html` render a 3D bar chart using a Gaussian filter and gradients, respectively. In the latter code sample, the definition of a linear gradient and a radial gradient increase the code size by 30%, which gives you an indication that D3-based gradient definitions can be quite lengthy in relation to the D3 code that creates the graphics effect.

The HTML Web page `Multi3DBarChart1Filter1Frosted1.html` is another example of adding an SVG `<filter>` element and gradient element. The HTML Web page `Multi3DBarChart1Filter1Frosted1.html` further adds two SVG `<pattern>` elements to render yet another 3D bar chart with an interesting visual effect.

The HTML Web pages `Multi3DBarChart1QBezier1.html` and `Multi3DBarChart1CBezier1.html` render a 3D bar chart by drawing each bar element with a quadratic Bezier curve and a cubic Bezier curve, respectively.

The HTML Web pages `Multi3DBarChart1Cylinders1.html` and `Multi3DBarChart1Cylinders2.html` render a 3D bar chart by

drawing each bar element as a set of stacked ellipses and programmatically calculating the (R,G,B) values for each ellipse.

The HTML Web pages `Multi3DBarChart1Cones1.html` and `Multi3DBarChart1Cones2.html` render a 3D bar chart by drawing each bar element as a cone based on a set of shrinking stacked ellipses and programmatically calculating the (R,G,B) values for each ellipse.

The HTML Web pages `Multi3DBarChart1SineCylinders1.html` and `Multi3DBarChart1SineCylinders2.html` render a 3D bar chart by drawing each bar element as a sine-based set of stacked ellipses and programmatically calculating the (R,G,B) values for each ellipse. Keep in mind that the cone-based and cylinder-based code samples create a large number of SVG `<ellipse>` elements, and the performance delay is noticeable, so use these code samples judiciously.

The HTML Web page `Multi3DBarChart1Lines1.html` also renders a 3D bar chart, except the front face is rendered as a pair of triangles instead of a rectangle. In addition, each triangle is rendered using a set of SVG `<line>` segments, where the (R,G,B) color of each line segment is calculated programmatically in a loop instead of using built-in gradients.

The HTML Web page `Multi3DBarChart1Anim1.html` renders a 3D bar chart where clicking anywhere inside the bar chart generates an animation effect that uses the D3 methods `.transition()` and `.duration()` to rescale the bar chart each time users click on the bar chart.

One more observation: all of the code samples use a combination of D3 functionality and JavaScript code that are discussed in the first three chapters of this book. Feel free to use the techniques in these examples in code samples in other chapters of this book. Perhaps they can inspire you to create some of your own great visual effects!



Incidentally, the CD also contains a number of code samples in Chapters 2 and 3 that create multiple animation effects (using simple JavaScript) as soon as the HTML Web pages are loaded into a browser. Some of these animation samples contain fewer than one hundred lines of code, such as this sample in Chapter 2:

```
CardioidEllipses1Grad2Scale1AutoMultiAnim1.html
```

Finally, Mike Bostock created two very nice multiple-transition animation effects, which can be found at <http://bl.ocks.org/mbostock/1256572> and <http://bl.ocks.org/mbostock/1353700>. You can learn some interesting D3 techniques from reading the code in this pair of online code samples.

## SUMMARY

---

This chapter showed you how to use D3 to create various types of bar charts in D3, such as horizontal, vertical, and scaled bar charts. Then you learned how to render a bar chart with a 3D effect and how to render bar charts with filters in D3. You also saw how to handle mouse events and how to add animation effects to bar charts with D3. The next chapter shows you how to create other charts types (such as pie charts and line graphs) and how to work with other data types (such as CSV text files) in D3.



## OTHER CHART TYPES AND DATA FORMATS

Chapter 3 contains code samples exclusively for bar charts, and this chapter shows you how to use D3 to create other types of charts and graphs, such as line graphs, pie charts, and histograms.

The first part of this chapter shows you how to create a line graph followed by an example of a multiline graph. The second part of this chapter shows you how to use D3 to render various types of scatter charts, some of which involve mouse events. The third part of this chapter shows you how to create a pie chart and a histogram. The fourth part of this chapter contains code samples for rendering charts using data in various other formats such as CSV-based text files and JSON.

### RENDERING A LINE GRAPH

---

A line graph consists of a set of contiguous line segments, each of which you can easily construct using D3.

Listing 4.1 displays the contents of `LineGraph1.html` that illustrate how to create and render a line graph.

#### **LISTING 4.1: *LineGraph1.html***

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8" />
    <title>Line Graph</title>

    <script src="d3.min.js"></script>

    <style>
      svg {
        position: absolute;
```



```

    left: 40px; top: 10px;
  }
</style>
</head>

<body>
<script>
  var width=800, height=500, factor=3;
  var cRadius=5, cColor="blue", cStroke="black", cOp=0.7;

  var dataValues = [[10,80], [50, 150], [100,50],
                    [200,250], [250,200], [300,300]];

  var xScale = d3.scale.linear()
    .domain([0, d3.max(dataValues,
      function(d){return d[0]}))]
    .range([0, 600]);

  var yScale = d3.scale.linear()
    .domain([0, d3.max(dataValues,
      function(d){return d[1]}))]
    .range([300, 50]);

  var line = d3.svg.line()
    .x(function(d){return xScale(d[0]);})
    .y(function(d){return yScale(d[1]);})

  // create an SVG container...
  var svg = d3.select("body").append("svg")
    .attr("width", width)
    .attr("height", height);

  // create a line graph...
  svg.append("svg:path")
    .attr("class", "line")
    .attr("stroke", "#f00")
    .attr("stroke-width", 4)
    .attr("fill", "none")
    .attr("d", line(dataValues));

  // add circles at the vertices...
  svg.selectAll("circle")
    .data(dataValues)
    .enter()
    .append("circle")
    .attr("cx", function(d){return xScale(d[0]);})
    .attr("cy", function(d){return yScale(d[1]);})
    .attr("r", cRadius)
    .style("fill", cColor)
    .style("fill-opacity", cOp)
    .style("stroke", cStroke)
    .on("mouseover", function() {
      var r = d3.select(this).attr("r");
      d3.select(this).attr("r", factor*r);
    })
    .on("mouseout", function() {
      var r = d3.select(this).attr("r");

```

```

        d3.select(this).attr("r", r/factor);
    })
</script>
</body>

</html>

```

Listing 4.1 contains boilerplate code and a `<script>` element that defines various JavaScript variables. The JavaScript array `dataValue` contains pairs of numbers where each pair represents an endpoint of a line segment. Next, Listing 4.2 initializes variables that scale the x-values and the y-values of the endpoints of the line segments, followed by the definition of a variable that acts as a line function, as shown here:

```

var line = d3.svg.line()
    .x(function(d){return xScale(d[0]);})
    .y(function(d){return yScale(d[1]);})

```

The next portion of Listing 4.1 creates and then appends an SVG `<svg>` element to the `<body>` element, followed by `TMCHID3` that displays a circle over each vertex in the line graph. In addition, users can increase and decrease the radius of the circles during a mouseover and mouseout event, respectively, as shown here:

```

// add circles at the vertices...
svg.selectAll("circle")
    .data(dataValues)
    .enter()
    .append("circle")
    .attr("cx", function(d){return xScale(d[0]);})
    .attr("cy", function(d){return yScale(d[1]);})
    .attr("r", cRadius)
    .style("fill", cColor)
    .style("fill-opacity", cOp)
    .style("stroke", cStroke)
    .on("mouseover", function() {
        var r = d3.select(this).attr("r");
        d3.select(this).attr("r", factor*r);
    })
    .on("mouseout", function() {
        var r = d3.select(this).attr("r");
        d3.select(this).attr("r", r/factor);
    })

```

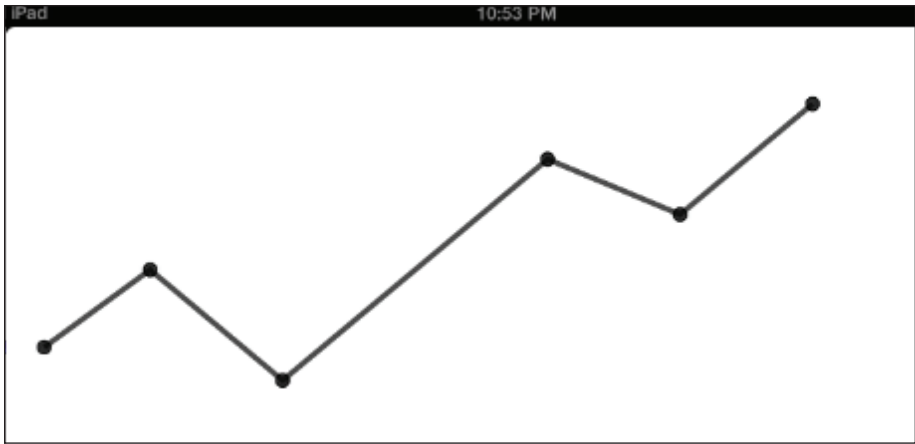


Notice that all the line segments in Listing 4.1 are rendered with the same color. The HTML Web page `LineGraph2.html` on the CD contains an example of rendering different line segments with different colors. The relevant section of code from `LineGraph2.html` is shown here:

```

// draw the line segments...
svg.selectAll("rect")
    .data(dataXValues)
    .enter()

```



**FIGURE 4.1** A Line Graph on an iPad3.

```
.append("line")
.attr("x1", function(d, i) {
    return i * (width/dataXValues.length);
})
.attr("y1", function(d, i) {
    return dataYValues[i];
})
.attr("x2", function(d, i) {
    return (i+1) * (width/dataXValues.length);
})
.attr("y2", function(d, i) {
    return dataYValues[i+1];
})
.style("stroke-width", lineWidth)
.style("stroke", function(d, i) {
    return lineColors[i%lineColors.length];
});
```

Figure 4.1 displays a line graph when Listing 4.1 is rendered on an iPad3.

## RENDERING MULTIPLE NONLINEAR GRAPHS

Now that you know how to render a simple line graph, the code sample in this section shows you how to define a multidimensional array of data values that represent a set of nonlinear graphs. The code is surprisingly easy, especially when you compare this code with a manual solution.

Listing 4.2 displays the contents of `MultiNonLinearGraphs1.html` that illustrate how to create and render multiple nonlinear graphs whose JSON-based data values are defined in a JavaScript array.

### **LISTING 4.2:** *MultiNonLinearLineGraphs1.html*

```
<!DOCTYPE html>
<html>
<head>
```

```

<meta charset="utf-8" />
<title>Multi-Line Graph</title>

<script src="d3.min.js"></script>

<style>
  svg {
    position: absolute;
    left: 40px; top: 10px;
  }
</style>
</head>

<body>
  <script>
    var width=600, height=400, multX = 200;
    var lineSegmentWidth = 100, lineWidth=2;
    var lineColors = ["red", "yellow", "blue", "green"];

    var dataValues = [
      [{x: 0, value: 20}, {x: 1, value: 40}, {x: 2, value: 80}],
      [{x: 0, value: 50}, {x: 1, value: 90}, {x: 2, value: 200}],
      [{x: 0, value: 30}, {x: 1, value: 20}, {x: 2, value: 50}],
      [{x: 0, value: 80}, {x: 1, value: 40}, {x: 2, value: 350}]
    ];

    var line = d3.svg.line()
      // .interpolate("monotone")
      // .interpolate("linear")
      .interpolate("basis")
      .x(function(d) {
        return multX*(d.x); })
      .y(function(d) {
        return d.value;
      });

    // create the <svg> element...
    var svg = d3.select("body")
      .append("svg")
      .attr("width", width)
      .attr("height", height);

    var group = d3.select("svg");

    group.selectAll(".line")
      .data(dataValues)
      .enter().append("path")
      .attr("class", "line")
      .attr("d", line)
      .style("stroke-width", lineWidth)
      .style("stroke", function(d, i) {
        return lineColors[i%lineColors.length];
      });
  </script>
</body>
</html>

```

The code in Listing 4.2 that is similar to Listing 4.1 is omitted, and the following discussion focuses on the two main differences between Listing 4.1 and Listing 4.2. The first key difference is the JavaScript array `dataValues`, which consists of numbers in a multidimensional array, as shown here:

```
var dataValues = [
  [{x: 0, value: 20}, {x: 1, value: 40}, {x: 2, value: 80}],
  [{x: 0, value: 50}, {x: 1, value: 90}, {x: 2, value: 200}],
  [{x: 0, value: 30}, {x: 1, value: 20}, {x: 2, value: 50}],
  [{x: 0, value: 80}, {x: 1, value: 40}, {x: 2, value: 350}]
];
```

As you can probably surmise, the preceding code snippet defines data values for four distinct graphs. However, the second key difference, which is not apparent simply by reading the code, is the fact that the Listing 4.2 renders nonlinear graphs. The nonlinear nature of the graphs is due to the fact that the definition of the JavaScript variable `line` involves the D3 `.interpolate()` method, as shown here:

```
var line = d3.svg.line()
  // .interpolate("monotone")
  // .interpolate("linear")
  .interpolate("basis")
  .x(function(d) {
    return multX*(d.x); })
  .y(function(d) {
    return d.value;
  });
```

The next portion of code in Listing 4.2 creates and appends an SVG `<svg>` element to the DOM, followed by the creation of the four nonlinear graphs using `TMCHID3`, as shown here:

```
var group = d3.select("svg");

group.selectAll(".line")
  .data(dataValues)
  .enter().append("path")
  .attr("class", "line")
  .attr("d", line)
  .style("stroke-width", lineWidth)
  .style("stroke", function(d, i) {
    return lineColors[i%lineColors.length];
  });
```

Experiment with Listing 4.2 to observe the type of nonlinear graphs that are rendered when you specify the other strings in the D3 `.interpolate()` method.

Figure 4.2 displays a set of nonlinear graphs on an iPad3.



**FIGURE 4.2** A Set of Nonlinear Graphs on an iPad3.

## SCATTER CHARTS WITH AXES AND MOUSE EVENTS

Listing 4.3 displays the contents of `ScatterAxes1.html` that illustrate how to create a scatter chart with labeled horizontal and vertical axes.

### **LISTING 4.3:** *ScatterAxes1.html*

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>Scatter Chart with Axes</title>
    <script src="d3.min.js"></script>

    <style>
      .axes {
        stroke: black; fill: none;
        shape-rendering: crispEdges;
      }
    </style>
  </head>

  <body>
    <script>
      var width=500, height=300, scalePadding=30;
      var fontSize="12px", fontColor="red", index=0;
      var circleColors = ["red", "green", "blue"];
```

```

var tickCount=5, minWidth=10, maxWidth=20;

var dataValues = [
    [75, 20], [250, 90], [350, 50], [120, 30],
    [400, 25], [450, 60], [125, 67], [185, 20]
];

// create an SVG element...
var svg = d3.select("body")
    .append("svg")
    .attr("width", width)
    .attr("height", height);

// create scale functions...
var xScale = d3.scale.linear()
    .domain([0, d3.max(dataValues, function(d) {
        return d[0]; })])
    .range([scalePadding, width-scalePadding*2]);

var yScale = d3.scale.linear()
    .domain([0, d3.max(dataValues, function(d) {
        return d[1]; })])
    .range([height - scalePadding, scalePadding]);

var rScale = d3.scale.linear()
    .domain([0, d3.max(dataValues, function(d) {
        return d[1]; })])
    .range([minWidth, maxWidth]);

// define the x axis...
var xAxis = d3.svg.axis()
    .scale(xScale)
    .orient("bottom")
    .ticks(tickCount);

// define the y axis
var yAxis = d3.svg.axis()
    .scale(yScale)
    .orient("left")
    .ticks(tickCount);

// create the 'scatter' rectangles...
svg.selectAll("rect")
    .data(dataValues)
    .enter()
    .append("rect")
    .attr("fill", function(d) {
        index = d[0]%circleColors.length;
        return circleColors[index];
    })
    .attr("x", function(d) {

```

```

        return xScale(d[0]);
    })
    .attr("y", function(d) {
        return yScale(d[1]);
    })
    .attr("width", function(d) {
        return rScale(d[1]);
    })
    .attr("height", function(d) {
        return rScale(d[1]);
    })
    .on("mouseover", function(d) {
        d3.select(this).attr("fill", "black");
    })
    .on("mouseout", function(d) {
        // restore original color
        index = d[0] % circleColors.length;
        d3.select(this).attr("fill", circleColors[index]);
    });

// render the x axis...
svg.append("g")
    .attr("class", "axes")
    .attr("transform",
        "translate(0," + (height - scalePadding) + ")")
    .call(xAxis);

// render the y axis...
svg.append("g")
    .attr("class", "axes")
    .attr("transform",
        "translate(" + scalePadding + ",0)")
    .call(yAxis);
</script>
</body>
</html>

```

Listing 4.3 also contains code that is similar to Listing 4.1, which will not be repeated here. However, Listing 4.3 defines three JavaScript variables for scaling the data values for the horizontal axis, the vertical axis, and the values for the width and height of each rectangle that represents a scatter point. The definition for the scaling variables is compact and dense in terms of the functionality. For example, consider the definition of the JavaScript variable `rScale` that is shown here:

```

var rScale = d3.scale.linear()
    .domain([0, d3.max(dataValues, function(d) {
        return d[1]; })])
    .range([minWidth, maxWidth]);

```

The preceding code snippet performs a linear scaling of the set of numbers between 0 and the maximum value of the second component of the



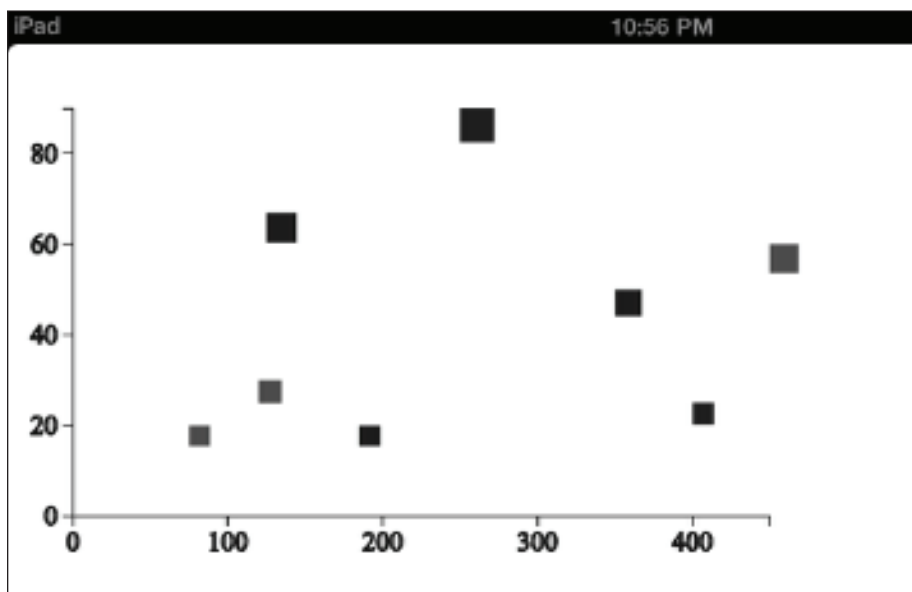
number pairs in the `dataValues` array. The result from the preceding step undergoes another scaling operation to ensure that it is scaled to a number between `minWidth` and `maxWidth` (both of which are defined earlier in Listing 4.3). In case you don't fully appreciate the functionality of the preceding code snippet, try to reproduce the same functionality using plain JavaScript!

The next portion of code in Listing 4.3 uses `TMCIID3` to create SVG `<rect>` elements that represent the points in the scatter plot, which involves setting values for the `x`, `y`, `width`, and `height` attributes of each rectangle. This code block also sets the color of a rectangle to black during a `mouseover` event, and then sets the color of the rectangle to its initial color during a `mouseout` event, as shown here:

```
.on("mouseover", function(d) {
    d3.select(this).attr("fill", "black");
})
.on("mouseout", function(d) {
    // restore original color
    index = d[0] % circleColors.length;
    d3.select(this).attr("fill", circleColors[index]);
});
```

Notice the use of the keyword `this` in the preceding code block to reference the rectangle that is involved in the mouse-related events.

Figure 4.3 displays a scatter chart with axes on an iPad3.



**FIGURE 4.3** A Scatter Chart with Axes on an iPad3.

## SELECTING EQUAL DATA POINTS IN SCATTER CHARTS

Listing 4.4 displays part of `ScatterAxes1EqualDataPoints1.html` that illustrates how to display only the data points that have equal size (in terms of the value of the radius) whenever users hover over any given data point. For instance, if users hover over a data point whose circle has radius 10, then all other circles with radius 10 (if any) are highlighted with opacity 1, and all other circles with a different radius are assigned a much smaller opacity value to give them a dimmed appearance. When users move their mouse away from the given data point all the circles are restored to their initial appearance (i.e., they have opacity 1 with a fill color of blue).

The code in Listing 4.4 shows you how to achieve this effect, which involves another new technique: TMCHID3 code block that contains TMCHID3 used in a nested fashion. Note that only the new code is shown in Listing 4.4, but the entire source code is available in the CD.



### LISTING 4.4: *ScatterAxes1EqualDataPoints1.html*

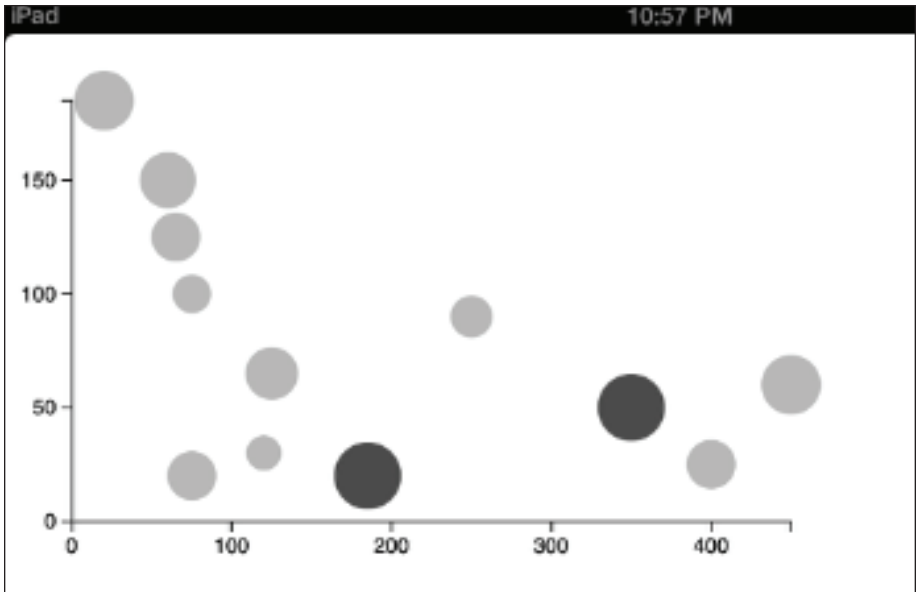
```
<!DOCTYPE html>
// details omitted for brevity
.on('mouseover', function(d,i){
    // get the radius of the current circle
    moRadius = d3.select(this).attr("r");

    // compare to the radius of the other circles
    d3.select("body")
      .selectAll("circle")
      .attr("fill", mouseOverColor)
      .attr("opacity", function(d) {
        currRadius = d3.select(this).attr("r");

        if(currRadius == moRadius) {
            opacity = maxOpacity;
        } else {
            opacity = minOpacity;
        }

        return opacity;
      })
})
.on('mouseout', function(d,i){
    // make all circles visible again...
    d3.select("body")
      .selectAll("circle")
      .attr("fill", circleColor)
      .attr("opacity", 1)
})
// details omitted for brevity
```

Listing 4.4 shows you the code for the `mouseover` event, which starts by selecting the radius of the current circle (once again, note the use of the `this`



**FIGURE 4.4** A Scatter Chart on an iPad3 with Equal Data Points Highlighted.

keyword). The next block of code selects the set of circles whose radius is equal to the radius of the circle associated with the `mouseover` event. Moreover, this code block sets the `opacity` attribute to `maxOpacity` for the matching circles and sets the opacity of all the other circles to `minOpacity`. During the `mouseout` event the color of all the circles is set to their initial value.

Figure 4.4 displays a scatter chart on an iPad3 where equal data points are highlighted.

## RENDERING PIE CHARTS

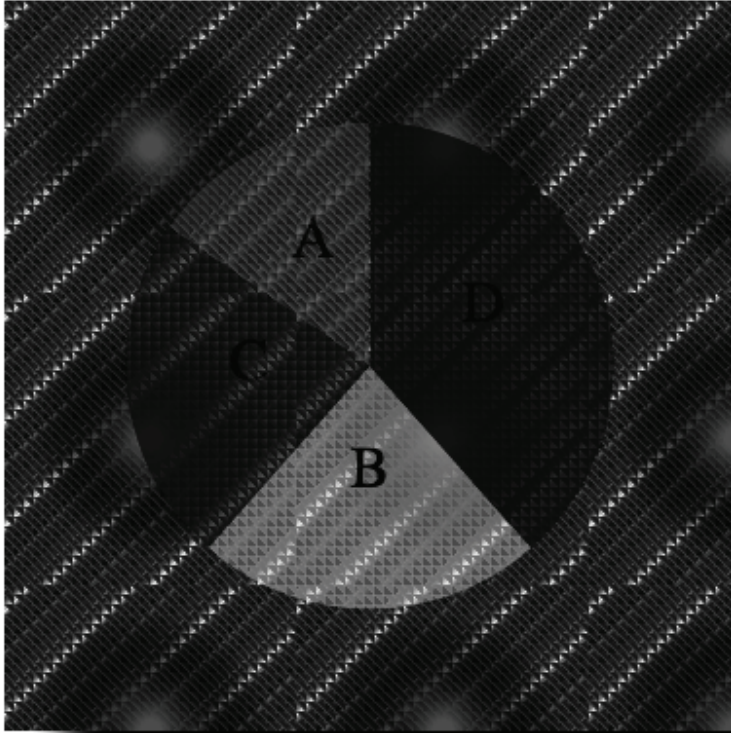
Another graphic is the pie chart, which probably rivals bar charts in terms of popularity. A nice online example of creating a pie chart is available here at <https://gist.github.com/1203641>. You can read the HTML Web page in this Website, which uses the D3 methods `.arc()`, `.pie()`, and `arc.centroid()` to create the desired effect. The modest set of modifications to the preceding code sample include:

- Adding a CSS3 stylesheet with gradient effects
- Changing the text font size during a hover event
- Changing the opacity of a wedge during a hover event
- Centering the pie chart

Figure 4.5 displays a pie chart in a Chrome browser with the modified pie-chart code.



See the next-to-last section of this chapter for a description of other pie-related charts on the CD.



**FIGURE 4.5** A Pie Chart in a Chrome Browser with Hover Effects.

## A HISTOGRAM WITH ANIMATION EFFECTS

Although histograms are basically the same as bar charts in terms of the information that they can represent, D3 enables you to create a nice animation effect on histograms.

When you launch the HTML Web page in this section, you will initially see all the bar elements of the histogram displayed with zero height (starting from the bottom of the screen). Each bar element will then grow upward until it reaches its terminal height, thereby creating an animation effect.

Listing 4.5 displays the contents of `HistogramAnimation1.html` that illustrate how to render a histogram with an initial animation effect followed by a scaling effect when users move their mouse over the histogram.

### **LISTING 4.5: *HistogramAnimation1.html***

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8" />
```

```

<title>Histogram Animation</title>
<script src="d3.min.js"></script>
</head>

<body>
  <script>
    var width=400, height=400;
    var sampleCount=1000, randl=0, minValue=10;
    var duration = 4000, dataValues = [];
    var barColors = ["red", "yellow", "green", "blue"];

    for(var i = 0; i < sampleCount; i++) {
      randl = minValue+50*Math.random();
      dataValues.push(randl);
    }

    var histogram = d3.layout.histogram()(dataValues);

    var x = d3.scale.ordinal()
      .domain(histogram.map(function(d) { return d.x; }))
      .rangeRoundBands([0, width]);

    var y = d3.scale.linear()
      .domain([0, d3.max(histogram, function(d) {return d.y;})])
      .range([0, height]);

    var vis = d3.select("body").append("svg:svg")
      .attr("width", width)
      .attr("height", height)
      .append("svg:g")
      .attr("transform", "translate(.5)");

    vis.selectAll("rect")
      .data(histogram)
      .enter().append("svg:rect")
      .attr("transform", function(d) {
        return "translate("+x(d.x)+","+ (height - y(d.y))+")";
      })
      .attr("width", x.rangeBand())
      .attr("y", function(d) { return y(d.y); })
      .attr("height", 0)
      .attr("fill", function(d) {
        return barColors[d.y % barColors.length]
      })
      .transition()
      .duration(duration)
      .attr("y", 0)
      .attr("height", function(d) { return y(d.y); });

    vis.append("svg:line")
      .attr("x1", 0)
      .attr("x2", width)
      .attr("y1", height)
      .attr("y2", height);
  </script>
</body>
</html>

```

Listing 4.5 contains boilerplate code followed by a `<script>` element that initializes some JavaScript variables. A simple loop initializes the height of each bar element, as shown here:

```
for(var i = 0; i < sampleCount; i++) {
  rand1 = minValue+50*Math.random();
  dataValues.push(rand1);
}
```

The next code snippet shows you how to use the D3 `.histogram()` method to indicate that we want to treat the numbers in the JavaScript array `dataValues` as elements of a histogram:

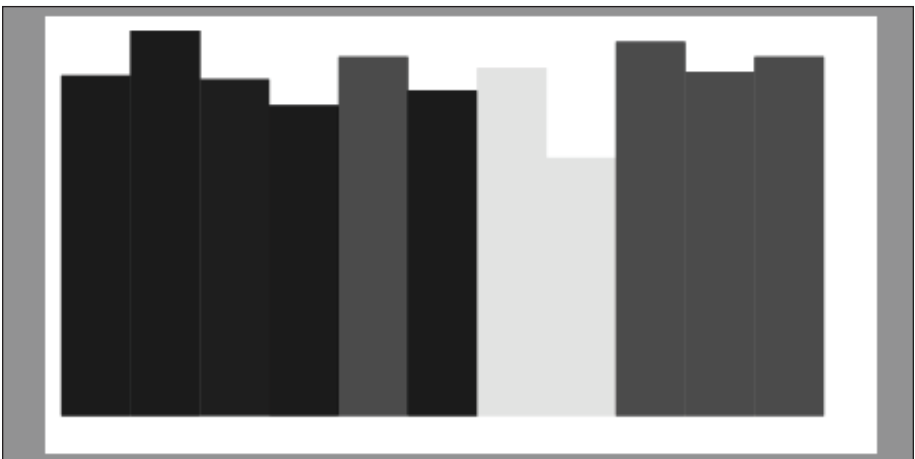
```
var histogram = d3.layout.histogram()(dataValues);
```

The next portion of code defines some scaling-related variables then creates and appends an SVG `<svg>` element to the DOM. Next, the code uses `TMCID3` to render a set of bar elements, which you have seen many times. The animation effect in the histogram is created with this simple code snippet:

```
.transition()
  .duration(duration)
  .attr("y", 0)
  .attr("height", function(d) { return y(d.y); });
```

As you can see, the preceding code snippet uses the D3 `.transition()` method that invokes the `.duration()` method, which specifies the value of the JavaScript variable `duration` (which is set to 4,000 milliseconds) as the duration of the animation effect.

Figure 4.6 displays a screenshot of an animated histogram in a Chrome browser on a Macbook Pro.



**FIGURE 4.6** An Animated Histogram in a Chrome Browser on a Macbook Pro.

## WORKING WITH OTHER DATA FORMATS AND DATA FILES

D3 provides methods for handling data in other formats including CSV, JSON, and XML. Because SVG documents are also XML documents, you can also load SVG documents into an existing HTML Web page.

Before delving into the D3 methods, keep in mind that there are other ways you can retrieve data in an HTML5 Web page. In addition to using pure JavaScript and jQuery, you could also use Yahoo! Query Language (YQL) or similar code in conjunction with Yahoo! Pipes to retrieve data and then integrate that YQL code with your D3 code in the same HTML5 Web page.

With these points in mind, we'll briefly look at two techniques for retrieving data, both of which are optional, so you can skim through the code (or skip entirely) without any loss of continuity.

### The XMLHttpRequest Request Object

This section contains an example of pure JavaScript for making requests for files. Listing 4.6 is for illustration purposes so you can compare the code in Listing 4.6 with the code samples later in this chapter that use D3 methods. In the (albeit unlikely) event that you do need to use pure JavaScript code in your HTML Web pages, you can find articles with detailed information on the Internet.

Listing 4.6 shows you how to invoke an XMLHttpRequest request object to read an XML document on the file system.

#### **LISTING 4.6: Using the XMLHttpRequest Object**

```
<script>
  var xmlHTTP, myFile = "http://localhost:8080/sample.xml";
  var url = "http://localhost:8080/sample.xml";

  function loadXML(url, callback) {
    if (window.XMLHttpRequest) {
      // Chrome, Firefox, IE7+, Opera, and Safari
      xmlHTTP = new XMLHttpRequest();
    } else {
      // IE5 and IE6
      xmlHTTP = new ActiveXObject("Microsoft.XMLHTTP");
    }

    xmlHTTP.onreadystatechange = callback;
    xmlHTTP.open("GET", url, true);
    xmlHTTP.send();
  }

  function init() {
    loadXML(myFile, function() {
      if(xmlHTTP.readyState==4 && xmlHTTP.status==200) {
        document.getElementById("myDiv").innerHTML =
          xmlHTTP.responseText;
      }
    });
  }
</script>
```

Listing 4.6 starts with the `loadXML()` function that is invoked by the JavaScript `init()` function, which is defined later in the code sample. As you can see, `loadXML()` contains logic for initializing the `xmlHTTP` variable followed by code that specifies the callback function `callback` and the GET method. Returning to the `init()` method, you can see a conditional statement that determines when the data is actually available, after which the `myDiv` element (not shown in the code) is populated with the returned data.

As mentioned at the beginning of this section, Listing 4.6 is only for illustrative purposes, and it will probably give you an appreciation for the D3 methods that you will learn about later in this chapter. The next section shows you a simple code snippet using the jQuery `.ajax()` method that makes an invocation to retrieve data.

## The jQuery .ajax() Method

Although we have not discussed jQuery in this book, you can probably understand the following code sample that illustrates how to use the jQuery `.ajax()` method to retrieve an XML document:

```
$.ajax({
  url: url,
  type: "get",
  success: GotData,
  dataType: 'xml'
});
```

Clearly, the preceding code block is much simpler than the code in Listing 4.6. In the preceding code block, you also need to define a JavaScript function called `GotData()` where you would process the result of the Ajax invocation. Presumably, you would extract the individual rows of the result set and populate a JavaScript array (or object) with the data in the result set.

Fortunately, D3 provides useful methods for making requests and parsing data in the same code block, which you will see in the following sections.

## Useful D3 Methods for CSV Files

D3 provides aptly named methods whose names are indicative of the data formats for which they are designed. Specifically, the D3 methods for working with CSV text files are `d3.text()`, `d3.csv()`, `d3.json()`, and `d3.xml()`. The `d3.text()` method handles CSV-formatted data consisting of pure data, whereas the `d3.csv()` method handles CSV-formatted data whose first line provides labels for the data fields.

The rest of this chapter provides various code samples that show you how to manipulate data with some of these formats in D3.



## CSV: Synchronous Versus Nonsynchronous D3 Methods

One important point to remember about the difference between several CSV-based D3 methods is explained by Mike Bostock:

Unlike `d3.csv`, `d3.csv.parse` and `d3.csv.parseRows` are synchronous, so there's no callback required. The return value (data above) is ready for use immediately after you call `d3.csv.parseRows`. The only asynchronous part is loading the file via `d3.text`.

## D3 and External Resources

In addition to D3 support for the preceding data types, D3 provides built-in functionality to load in the following types of external resources:

- a JSON blob
- an HTML document fragment
- an XML document fragment
- a tab-separated values (TSV) file

Because this book does not contain examples of handling the preceding list of external resources, you can check for code samples in online forums.

## LINE GRAPHS WITH CSV DATA AND MOUSE EVENTS

The code sample in this section shows you how to load data from a CSV file with one thousand rows of data, where each row represents the four values of a line segment (two values for each endpoint).

This code sample illustrates the following concept: whenever users move their mouse over a line segment, the endpoints of that line segment are appended as a JSON-formatted string to an array. Then when users click on the button, the one-thousand-line segments are removed and replaced with the line segments that users selected via a mouseover event.

This code sample also accesses data from a text file on the file system, so you must use a Web server instead of a file manager to load the HTML Web page into a browser.

Listing 4.7 displays the contents of `CSVOneK.html` that illustrate how to perform the functionality described above.

### LISTING 4.7: *CSVOneK.html*

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8" />
    <title>Loading 1K CSV Elements</title>
    <script src="d3.min.js"></script>
```

```

<style>
  p {
    position: absolute;
    width: 300px;
    left: 5px; top: 0px;
    font-size: 16px;
  }

  button {
    position: absolute;
    width: 100px;
    left: 300px; top: 20px;
  }

  svg {
    position: absolute;
    left: 0px; top: 50px;
  }
</style>
</head>

<body>
  <div>
    <p>Move Your Mouse Over The Line Segments</p>
    <button id="button" width="200"
      onclick="showMouseOverLines()">Click Me</button>
  </div>

  <script>
    var width=600, height=350, lineWidth=2;
    var lineColors = ["red", "yellow", "blue"];
    var counter=0, index=0, radius=4;
    var mox1=0, moy1=0, mox2=0, moy2=0;
    var mouseArray = [];

    var svg = d3.select("body")
      .append("svg")
      .attr("id", "svg")
      .attr("width", width)
      .attr("height", height);

    d3.csv("CSVOneK.csv", function(lines) {
      lines.forEach(function(d) {
        // set the values for both endpoints
        d.x1 = +d.x1;
        d.y1 = +d.y1;
        d.x2 = +d.x2;
        d.y2 = +d.y2;

        // append each line segment here
        svg.append("line")
          .attr("x1", d.x1)
          .attr("y1", d.y1)
          .attr("x2", d.x2)
          .attr("y2", d.y2)
          .style("stroke-width", lineWidth)
          .style("stroke", function(d, i) {

```

```

        ++counter;
        index = counter % lineColors.length;
        return lineColors[index];
    })
    .on("mouseover", function(d) {
        // append the line segment to the array...
        mox1 = d3.select(this).attr("x1");
        moy1 = d3.select(this).attr("y1");
        mox2 = d3.select(this).attr("x2");
        moy2 = d3.select(this).attr("y2");

        mouseArray.push({ "mox1":mox1, "moy1":moy1,
                          "mox2":mox2, "moy2":moy2 });

        d3.select(this).attr("stroke","black")
            .attr("stroke-width",10);
    });

    svg.append("circle")
        .attr("cx", d.x1)
        .attr("cy", d.y1)
        .attr("r", radius)
        .style("fill", function(d, i) {
            ++counter;
            index = counter % lineColors.length;
            return lineColors[index];
        })
        .on("mouseover", function(d, i) {
            d3.select(this).attr("fill","black")
                .attr("stroke","green")
                .attr("stroke-width",4)
                .attr("r",2*radius)
        });
    });
});

function showMoveOverLines() {
    // delete the lines...
    svg.selectAll("line").remove();

    // delete the circles...
    svg.selectAll("circle").remove();

    counter = 0;
    svg.selectAll("line")
        .data(mouseArray)
        .enter().append("line")
        .attr("x1", function(d, i) { return d["mox1"] })
        .attr("y1", function(d, i) { return d["moy1"] })
        .attr("x2", function(d, i) { return d["mox2"] })
        .attr("y2", function(d, i) { return d["moy2"] })
        .style("stroke-width", 2*lineWidth)
        .style("stroke", function(d, i) {
            ++counter;
            index = counter % lineColors.length;
            return lineColors[index];
        });
}

```

```

    });
  }
</script>
</body>
</html>

```

Listing 4.7 starts with initialization code that contains a simple `<style>` element for styling paragraphs, buttons, and the SVG `<svg>` element. The next portion of code contains a `<script>` element that initializes some JavaScript variables including the variable `mouseArray` that keeps track of the line segments that have been “moused over” by users.

After creating and appending an SVG `<svg>` element to the DOM, Listing 4.7 contains D3 code to read the data values in the CSV-formatted file `CSVOneK.csv`, which treats each row as the two endpoints of a line segment, as shown here:

```

d3.csv("CSVOneK.csv", function(lines) {
  lines.forEach(function(d) {
    // set the values for both endpoints
    d.x1 = +d.x1;
    d.y1 = +d.y1;
    d.x2 = +d.x2;
    d.y2 = +d.y2;

    // append each line segment here
    svg.append("line")

    // more code here
  })
}

```

The SVG `<svg>` element also specifies a `mouseover` event that appends the current line segment to the JavaScript array `mouseArray`, as shown here:

```

.on("mouseover", function(d) {
  // append the line segment to the array...
  mox1 = d3.select(this).attr("x1");
  moy1 = d3.select(this).attr("y1");
  mox2 = d3.select(this).attr("x2");
  moy2 = d3.select(this).attr("y2");

  mouseArray.push({ "mox1":mox1, "moy1":moy1,
                    "mox2":mox2, "moy2":moy2 });

  d3.select(this).attr("stroke","black")
                .attr("stroke-width",10);
});

```

The final portion of code in the `d3.csv()` code block uses a simple loop to add a set of circles to each end point of the line segments that are displayed.

The only other code block is the JavaScript array `showMoveOverLines()` that is executed when users click on the button that is displayed. This function

starts by removing all the lines and circles on the screen, and then it displays the line segments that have been “moused over” by users, as shown here:

```
function showMoveOverLines() {
    // delete the lines...
    svg.selectAll("line").remove();

    // delete the circles...
    svg.selectAll("circle").remove();

    // display the "moused over" line segments
};
```

Listing 4.8 displays the contents of the Perl script `CSVOneK.pl` that generates the CSV-formatted data in the text file `CSVOneK.csv` that is used in Listing 4.7.

#### **LISTING 4.8: *CSVOneK.pl***

```
#!/usr/bin/perl
use strict;
use warnings;

my $count = 0;
my $shape = 0;
my $range = 400;
my $maxCount = 1000;

# print header line:
print "x1,y1,x2,y2\n";

for ($count = 0; $count < $maxCount; $count++) {
    my $rand1 = int(rand($range));
    my $rand2 = int(rand($range));
    my $rand3 = int(rand($range));
    my $rand4 = int(rand($range));

    print $rand1 . "," . $rand2 . "," .
          $rand3 . "," . $rand4 . "\n";
}
```

Listing 4.8 contains a simple loop that generates four random numbers, each of which is between 0 and 400, and then prints those four comma-separated numbers as a single row.

You can launch the Perl script in Listing 4.11 and redirect the output to a text file `CSVOneK.csv` as shown here:

```
perl CSVOneK.pl > CSVOneK.csv
```

Obviously, you can modify the parameters as you wish in Listing 4.11 to generate other sets of numbers. You can also perform an Internet search and find lots of online articles that discuss Perl in greater detail.

## BAR CHARTS WITH THREE-DIMENSIONAL EFFECTS FROM COMMA-SEPARATED-VALUE FILES

In the preceding section you learned how to read data from a CSV file, and in Chapter 3 you saw how to create a multilevel 3D bar chart using randomly generated data values. Specifically, Listing 3.3 in chapter 3 shows you the relevant portion of the HTML Web page `Multi3DBarChart1CSV1.html` that populates a multidimensional array with data from a CSV file instead of using randomly generated values. The key points to notice are provided as comments to the sections of code.



The entire code listing for `Multi3DBarChart1CSV1.html`, as well as the Perl script `Multi3DBarChart1CSV1.pl`, is available on the CD. This Perl script creates the CSV data file `Multi3DBarChart1CSV1.csv` for Listing 4.9 by generating a set of random numbers based on hard-coded values that are closely coupled to the code in the HTML Web page. However, you can easily replace the CSV data file with actual data from your own sources.

### ***LISTING 4.9: Initializing a Multidimensional Array in Multi3DBarChart1CSV1.html***

```
// initialize bar heights from CSV file...
d3.text("Multi3DBarChart1CSV1.csv", function(text) {
  d3.csv.parseRows(text).map(function(row) {
    row.map(function(rowOfData) {
      // create an array of numbers...
      var tempRow = rowOfData.split(" ");

      // remove the trailing comma...
      tempRow = tempRow.splice(0, tempRow.length-1);

      // prepare for a new row of numbers...
      multiBarHeights[currRowIndex] = new Array(tempRow.length);

      // notice the "+" sign to convert to a number...
      for(var d=0; d<tempRow.length; d++) {
        multiBarHeights[currRowIndex][d] = +tempRow[d];
      }
    });
    ++currRowIndex;
  })

  // Since the d3.text() method is asynchronous, the
  // multi-dimensional array is not populated before
  // this point, so now you can render the bar chart
  render3DBarChart();
});
```

The code in Listing 4.9 is the counterpart to the following block of code that is in the HTML Web page `Multi3DBarChart1.html` in chapter 3:

```
for(var row=0; row<rowCount; row++) {
    multiBarHeights[row] = new Array(barCount);

    for(var b=0; b<barCount; b++) {
        barHeights[b] =
            Math.floor((maxHeight-minHeight)*Math.random()+
                        minHeight);

        multiBarHeights[row][b] = barHeights[b];
    }
}
```

The preceding code block initializes a two-dimensional array consists of `rowCount` rows of data where each row of data contains `barCount` data values. Each data value represents the height of a bar element. As you can see, the data values are randomly generated numbers that are calculated with the `Math.random()` function.

Listing 4.10 displays the contents of the Perl script `Multi3DBarChart CSV1.pl` that generates the data in the CSV data file `Multi3DBarChart CSV1.csv`. This Perl script contains very similar logic as the Perl script `CSVOneK.pl` that you saw earlier in this chapter.

#### **LISTING 4.10: *Multi3DBarChart1CSV1.pl***

```
#!/usr/bin/perl
use strict;
use warnings;

my $count      = 0;
my $shape      = 0;
my $range      = 0;
my $bar        = 0;
my $row        = 0;
my $barCount   = 20;
my $rowCount   = 8;
my $minHeight  = 20;
my $maxHeight  = 120;

$range = ($maxHeight-$minHeight);

for($row=0; $row<$rowCount; $row++) {
    for($bar = 0; $bar < $barCount; $bar++) {
        my $rand1 = int(rand($range))+$minHeight;
        print $rand1 . " ";
    }

    print "\n";
}
```

As you can see, Listing 4.10 contains a nested loop that generates eight rows of data, where each row contains twenty randomly generated numbers.

## ADDITIONAL CODE SAMPLES ON THE CD

---



The HTML Web page `Multi3DAreaChart1.html` shows you how to render a very rudimentary multilevel area chart. The Web pages `PieChart1Filter1.html` and `PieChart1Filter2.html` show you how to add a Gaussian blur filter to a pie chart. The first code sample contains a single blur filter, whereas the second code sample contains three Gaussian blur filters and one turbulence filter. After you have finished reading Chapter 5, you can experiment with these (as well as other) code samples by adding some of the additional types of filters that are described in Chapter 5.

The HTML Web page `D3PrimeFactors1.html` generates a scatter chart (although other types are also possible) that displays the prime factorization (including duplicates) for the integers between 2 and 100 (which you can increase as well). Prime numbers are displayed in red, and composite numbers are displayed in black.



The HTML Web Page `LineGraph1RandomPoints1.html` on the CD shows you how to render a line graph in D3 whose data points consist of randomly generated values. You can use this code sample when you need to simulate frequently changing data (such as the data that you see in a polygraph test).



The HTML Web Page `MovingLine1.html` on the CD shows you how to render a vertical line segment that follows your mouse as you move around the screen.

## SUMMARY

---

The first part of this chapter showed you how to use D3 to create simple line graphs, multiline graphs, scatter charts, and pie charts. Then you learned how to render a histogram with animation effects. Next, you saw how to add animation effects to a line graph. In the second part of this chapter, you learned how to work with other data formats, including CSV and JSON, and how to render various types of graphs based on data in these formats. The next chapter contains an overview of SVG with code samples that show you how to create basic 2D shapes in SVG.





## SVG ESSENTIALS

This chapter gives you an overview of SVG with SVG code samples that illustrate how to render various 2D shapes. SVG is an XML-based technology with support for linear gradients, radial gradients, filter effects, transforms (translate, scale, skew, and rotate), and animation effects using an XML-based syntax. Although SVG does not support 3D effects (which are available in CSS3), SVG provides some functionality that is unavailable in CSS3, such as support for arbitrary polygons, elliptic arcs, and quadratic and cubic Bezier curves.

The screenshots in this chapter of Android mobile applications on an Asus Prime Android tablet explicitly use Android 4.0.3. If you plan on developing Android-based mobile applications with SVG, keep in mind that although Android 4.0.3 does support most SVG features, there is no support for SVG filters. In addition, earlier versions of Android might not support other features that you want to use in your applications, so be sure to test your SVG-based mobile applications on a variety of devices.

As you will soon discover, it's possible to reference SVG documents in CSS selectors via the CSS `url()` function, which means you can harness the power of SVG via CSS selectors. Moreover, you can create HTML Web pages that contain a combination of D3, CSS, and pure SVG. Chapter 6 provides more information about referencing SVG from CSS selectors.

After you have finished reading this chapter, you will know how to use SVG elements and the required attributes for creating a variety of 2D shapes. You will also understand how to convert an SVG document into its D3 counterpart (which frequently involves the D3 `.attr()` method).

---

**NOTE** *Some of the SVG code samples in the first half of this chapter are also accompanied by D3 code snippets that show you how to render shapes (such as line segments, rectangles, and polygons) that are illustrated in the*

*SVG code samples. You can then refer to D3 code samples in the first four chapters of this book to easily put together the D3 code for the SVG code samples in the second half of this chapter.*

## OVERVIEW OF SVG

This section of the chapter contains various examples that illustrate some of the 2D shapes and effects that you can create with SVG. This section gives you a compressed overview of SVG, and you can search the Internet for SVG books and online tutorials, as well as the following open source projects:

<https://github.com/ocampesato/svg-graphics>  
<https://github.com/ocampesato/svg-filters-graphics>

## Basic Two-Dimensional Shapes in SVG

This section shows you how to render line segments and rectangles in SVG documents. As a simple example, SVG supports a `<line>` element for rendering line segments, and its syntax looks like this:

```
<line x1="20" y1="20" x2="100" y2="150".../>
```

An SVG `<line>` element renders line segments that connect the two points  $(x_1, y_1)$  and  $(x_2, y_2)$ .

SVG supports a `<rect>` element for rendering rectangles, and its syntax looks like this:

```
<rect width="200" height="50" x="20" y="50".../>
```

The SVG `<rect>` element renders a rectangle whose width and height are specified in the `width` and `height` attributes. The upper left vertex of the rectangle is specified by the point with coordinates  $(x, y)$ .

Listing 5.1 displays the contents of `BasicShapes1.svg` that illustrates how to render line segments and rectangles.

### LISTING 5.1 *BasicShapes1.svg*

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20001102//EN"
"http://www.w3.org/TR/2000/CR-SVG-20001102/DTD/svg-20001102.dtd">

<svg xmlns="http://www.w3.org/2000/svg"
      xmlns:xlink="http://www.w3.org/1999/xlink"
      width="100%" height="100%">
  <g>
    <!-- left-side figures -->
    <line x1="20" y1="20" x2="220" y2="20"
          stroke="blue" stroke-width="4"/>

    <line x1="20" y1="40" x2="220" y2="40"
```

```

        stroke="red" stroke-width="10"/>

<rect width="200" height="50" x="20" y="70"
      fill="red" stroke="black" stroke-width="4"/>

<path d="M20,150 1200,0 10,50 1-200,0 z"
      fill="blue" stroke="red" stroke-width="4"/>

<!-- right-side figures -->
<path d="M250,20 1200,0 1-100,50 z"
      fill="blue" stroke="red" stroke-width="4"/>

<path d="M300,100 1100,0 150,50 1-50,50 1-100,0 1-50,-50 z"
      fill="yellow" stroke="red" stroke-width="4"/>
</g>
</svg>

```

The first SVG `<line>` element in Listing 5.1 specifies the color `blue` and a `stroke-width` (i.e., line width) of 4, whereas the second SVG `<line>` element specifies the color `red` and a `stroke-width` of 10.

Notice that the first SVG `<rect>` element renders a rectangle that looks the same (except for the color) as the second SVG `<line>` element, which shows that it's possible to use different SVG elements to render a rectangle (or a line segment).

The SVG `<path>` element is probably the most flexible and powerful element because you can create arbitrarily complex shapes based on a concatenation of other SVG elements. Later in this chapter you will see an example of how to render multiple Bezier curves in an SVG `<path>` element.

An SVG `<path>` element contains a `d` attribute that specifies the points in the desired path. For example, the first SVG `<path>` element in Listing 5.1 contains the following `d` attribute:

```
d="M20,150 1200,0 10,50 1-200,0 z"
```

This is how to interpret the contents of the `d` attribute:

- move to the absolute location point (20, 150)
- draw a horizontal line segment 200 pixels to the right
- draw a line segment by moving 10 pixels to the right and 50 pixels down
- draw a horizontal line segment by moving 200 pixels to the left
- draw a line segment to the initial point (specified by `z`)

Similar comments apply to the other two SVG `<path>` elements in Listing 5.1. One detail to keep in mind is that uppercase letters (`C`, `L`, `M`, and `Q`) refer to absolute positions, whereas lowercase letters (`c`, `l`, `m`, and `q`) refer to relative positions with respect to the element that is to the immediate left. Experiment with the code in Listing 5.1 by using combinations of lowercase and uppercase letters to gain a better understanding of how to create different visual effects.

Because you are probably going to work primarily with D3 instead of SVG, a quick summary of how to render a line segment, a rectangle, and a path in D3 using an `svg` variable that is defined earlier in the code is presented next.

A D3 line segment looks like:

```
svg.append("svg:line")
    .attr("x1", 10)
    .attr("y1", 20)
    .attr("x2", 200)
    .attr("y2", 300)
    .style("stroke", "blue")
    .style("stroke-width", 3);
```

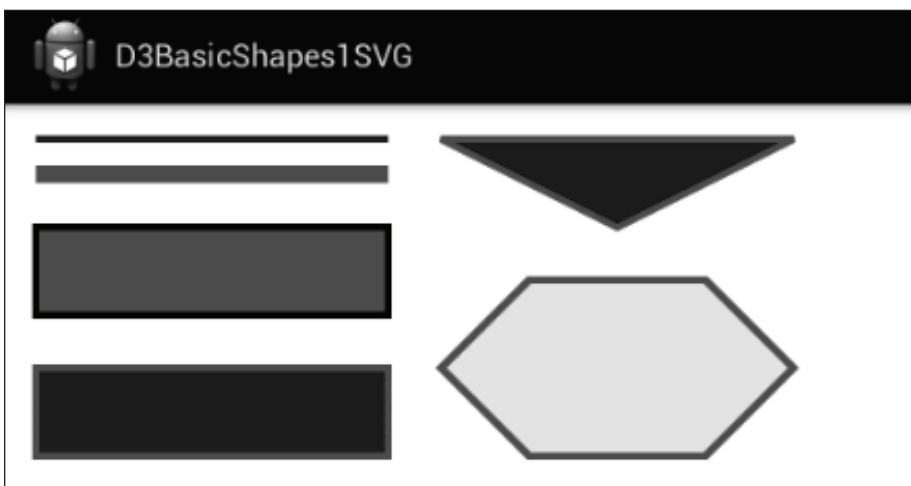
A D3 rectangle looks like:

```
svg.append("svg:rect")
    .attr("x", 10)
    .attr("y", 20)
    .attr("width", 200)
    .attr("height", 100)
    .attr("fill", "red");
```

A D3 path looks like:

```
svg.append("svg:path")
    .attr("d", "m0,0 10,-240 a200,200 0 0,0 0,240")
    .style("fill", "yellow")
```

Figure 5.1 displays the result of rendering the SVG document `BasicShapes1.svg` in a landscape-mode screenshot taken from an Android application running on an Asus Prime Android 4.0.3 10-inch tablet.



**FIGURE 5.1** SVG Lines and Rectangles on an Asus Prime Android 4.0.3 10-inch Tablet.

## SVG Gradients and the <path> Element

As you have probably surmised, SVG supports linear gradients as well as radial gradients that you can apply to 2D shapes. For example, you can use the SVG <path> element to define elliptic arcs (using the *d* attribute) and then specify gradient effects. The SVG <path> element contains a *d* attribute for specifying the shapes in a path, as shown here:

```
<path d="specify a list of path elements" fill="..." />
```

Listing 5.2 displays the contents of *BasicShapesLRG1.svg* that illustrate how to render 2D shapes with linear gradients and with radial gradients.

### LISTING 5.2 *BasicShapesLRG1.svg*

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20001102//EN"
"http://www.w3.org/TR/2000/CR-SVG-20001102/DTD/svg-20001102.dtd">

<svg xmlns="http://www.w3.org/2000/svg"
    xmlns:xlink="http://www.w3.org/1999/xlink"
    width="100%" height="100%">
  <defs>
    <linearGradient id="pattern1"
      x1="0%" y1="100%" x2="100%" y2="0%">
      <stop offset="0%" stop-color="yellow"/>
      <stop offset="40%" stop-color="red"/>
      <stop offset="80%" stop-color="blue"/>
    </linearGradient>

    <radialGradient id="pattern2">
      <stop offset="0%" stop-color="yellow"/>
      <stop offset="40%" stop-color="red"/>
      <stop offset="80%" stop-color="blue"/>
    </radialGradient>
  </defs>

  <g>
    <ellipse cx="120" cy="80" rx="100" ry="50"
      fill="url(#pattern1)"/>

    <ellipse cx="120" cy="200" rx="100" ry="50"
      fill="url(#pattern2)"/>

    <ellipse cx="320" cy="80" rx="50" ry="50"
      fill="url(#pattern2)"/>

    <path d="M 505,145 v -100 a 250,100 0 0,1 -200,100"
      fill="black"/>

    <path d="M 500,140 v -100 a 250,100 0 0,1 -200,100"
      fill="url(#pattern1)"
      stroke="black" stroke-thickness="8"/>
```

```

<path d="M 305,165 v 100 a 250,100 0 0,1 200,-100"
      fill="black"/>

<path d="M 300,160 v 100 a 250,100 0 0,1 200,-100"
      fill="url(#pattern1)"
      stroke="black" stroke-thickness="8"/>

<ellipse cx="450" cy="240" rx="50" ry="50"
         fill="url(#pattern1)"/>
</g>
</svg>

```

Listing 5.2 contains an SVG `<defs>` element that specifies a `<linearGradient>` element (whose `id` attribute has value `pattern1`) with three stop values using an XML-based syntax followed by a `<radialGradient>` element with three `<stop>` elements and an `id` attribute whose value is `pattern2`.

The SVG `<g>` element contains four `<ellipse>` elements, the first of which specifies the point (120, 80) as its center (`cx`, `cy`), with a major radius of 100, a minor radius of 50, and filled with the linear gradient `pattern1`, as shown here:

```

<ellipse cx="120" cy="80" rx="100" ry="50"
         fill="url(#pattern1)"/>

```

Similar comments apply to the other three SVG `<ellipse>` elements.

The SVG `<g>` element also contains four `<path>` elements that render elliptic arcs. The first `<path>` element specifies a black background for the elliptic arc defined with the following `d` attribute:

```

d="M 505,145 v -100 a 250,100 0 0,1 -200,100"

```

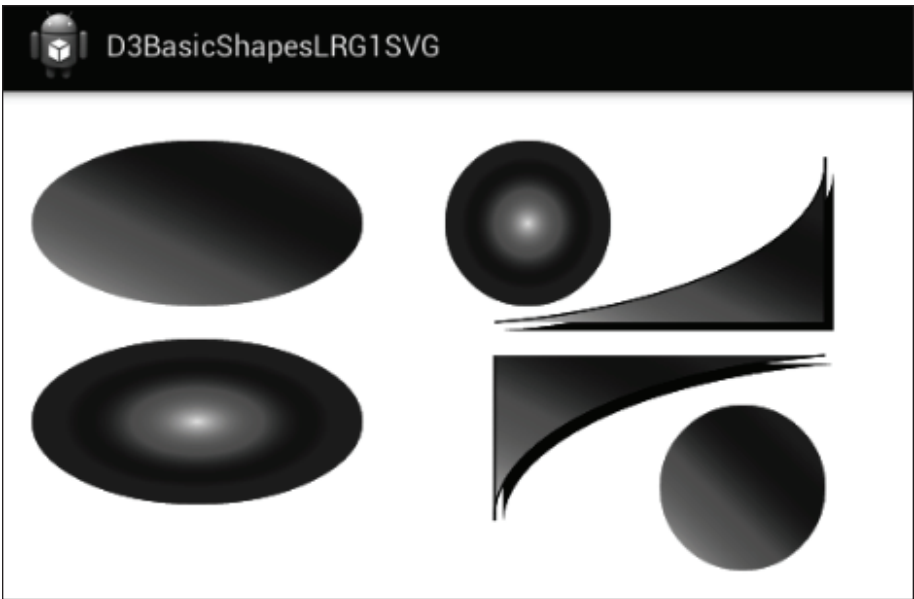
Unfortunately, the SVG syntax for elliptic arcs is nonintuitive, and it's based on the notion of major arcs and minor arcs that connect two points on an ellipse. This example is only for illustrative purposes, so we won't delve into a detailed explanation of how elliptic arcs are defined in SVG. However, the following Website provides additional details: <http://www.svgbasics.com/arcs.html>. Perform an Internet search to find additional links that discuss SVG elliptic arcs (and be prepared to spend some time experimenting with your own code samples).

The second SVG `<path>` element renders the same elliptic arc with a slight offset using the linear gradient `pattern1`, which creates a shadow effect. The code for an ellipse in D3 is:

```

svg.append("svg:ellipse")
  .append("ellipse")
  .attr("cx", 300)
  .attr("cy", 200)

```



**FIGURE 5.2** SVG Linear Radial Gradient Arcs on an Asus Prime Android 4.0.3 10-inch Tablet.

```
.attr("rx",      120)
.attr("ry",      80)
.attr("fill",    "red")
.attr("stroke",  "blue")
.attr("stroke-width", 2);
```



The code for a linear gradient and a radial gradient in D3 is somewhat lengthy (depending on the number of stop colors that you define), and you can see examples of both on the CD.

Similar comments apply to the other pair of SVG `<path>` elements, which render an elliptic arc with the radial gradient `pattern2` (also with a shadow effect).

Figure 5.2 displays the result of rendering `BasicShapesLRG1.svg`, in a landscape-mode screenshot taken from an Android application running on an Asus Prime Android 4.0.3 10-inch tablet.

## SVG `<polygon>` Element

The SVG `<polygon>` element contains a `polygone` attribute in which you can specify points that represent the vertices of a polygon. The SVG `<polygon>` element is most useful when you want to create polygons with an arbitrary number of sides, but you can also use this element to render line segments and rectangles. The syntax of the SVG `<polygon>` element looks like this:

```
<polygon path="specify a list of points" fill="..." />
```



Listing 5.3 displays the contents of a portion of `SvgCube1.svg` that illustrates how to render a cube in SVG.

### LISTING 5.3 *SvgCube1.svg*

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20001102//EN"
"http://www.w3.org/TR/2000/CR-SVG-20001102/DTD/svg-20001102.dtd">

<svg xmlns="http://www.w3.org/2000/svg"
      xmlns:xlink="http://www.w3.org/1999/xlink"
      width="100%" height="100%">
  <!-- <defs> element omitted for brevity -->

  <!-- top face (counter clockwise) -->
  <polygon fill="url(#pattern1)"
          points="50,50 200,50 240,30 90,30"/>

  <!-- front face -->
  <rect width="150" height="150" x="50" y="50"
        fill="url(#pattern2)"/>

  <!-- right face (counter clockwise) -->
  <polygon fill="url(#pattern3)"
          points="200,50 200,200 240,180 240,30"/>
</svg>
```

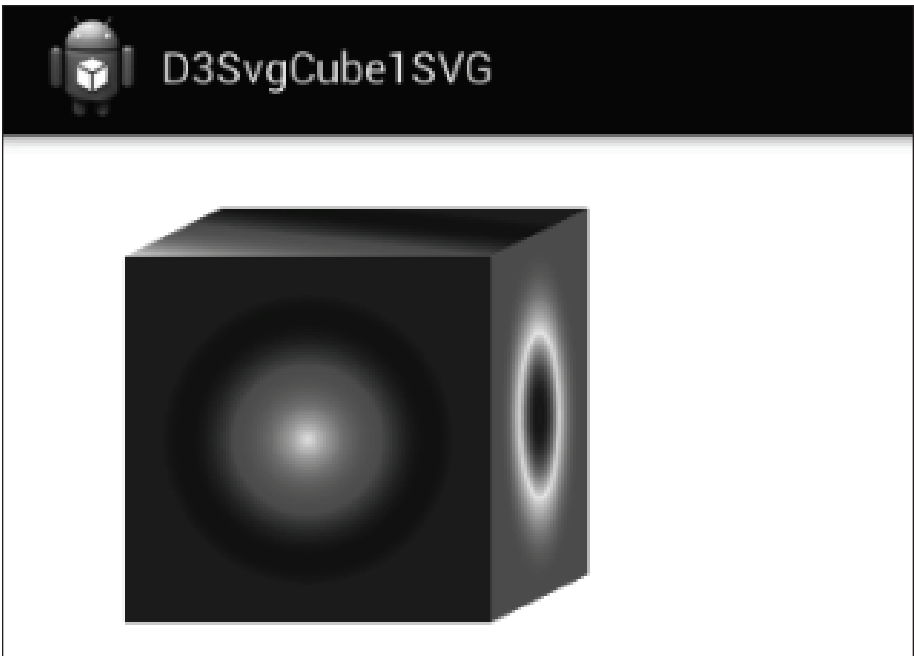
Listing 5.3 contains an SVG `<g>` element that contains the three faces of a cube, in which an SVG `<polygon>` element renders the top face (a parallelogram), an SVG `<rect>` element renders the front face, and another SVG `<polygon>` element renders the right face (also a parallelogram). The three faces of the cube are rendered with the linear gradient and the two radial gradients defined in the SVG `<defs>` element (not shown in Listing 5.3). The code for a blue polygon in D3 is:

```
var points2 = "200,50 200,200 240,180 240,30";
var polygon = gl.append("polygon")
    .style("fill", "blue")
    .attr("points", points2)
    .attr("stroke", "red")
    .attr("stroke-width", 1);
```

Figure 5.3 displays the result of rendering the SVG document `SvgCube1.svg` in a landscape-mode screenshot taken from an Android application running on an Asus Prime Android 4.0.3 10-inch tablet.

## BEZIER CURVES AND TRANSFORMS

SVG supports quadratic and cubic Bezier curves that you can render with linear gradients or radial gradients. You can also concatenate multiple Bezier curves using an SVG `<path>` element. Listing 5.4 displays the contents of `BezierCurves1.svg` that illustrate how to render various Bezier curves.



**FIGURE 5.3** An SVG Gradient Cube on an Asus Prime Android 4.0.3 10-inch Tablet.

**NOTE** *The transform-related effects are discussed later in this chapter.*

#### **LISTING 5.4** *BezierCurves1.svg*

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20001102//EN"
  "http://www.w3.org/TR/2000/CR-SVG-20001102/DTD/svg-20001102.dtd">

<svg xmlns="http://www.w3.org/2000/svg"
      xmlns:xlink="http://www.w3.org/1999/xlink"
      width="100%" height="100%">
  <!-- <defs> element omitted for brevity -->

  <g transform="scale(1.5,0.5)">
    <!-- scale a cubic Bezier curve with 'pattern1' -->
    <path d="m 0,50 C 400,200 200,-150 100,350"
          stroke="black" stroke-width="4"
          fill="url(#pattern1)"/>
  </g>

  <g transform="translate(50,50)">
    <!-- scale a red cubic Bezier curve -->
    <g transform="scale(0.5,1)">
      <path d="m 50,50 C 400,100 200,200 100,20"
            fill="red" stroke="black" stroke-width="4"/>
    </g>

    <!-- scale a yellow cubic Bezier curve -->
```

```

    <g transform="scale(1,1)">
      <path d="m 50,50 C 400,100 200,200 100,20"
        fill="yellow" stroke="black" stroke-width="4"/>
    </g>
</g>

<!-- translate/scale a blue cubic Bezier curve -->
<g transform="translate(-50,50)">
  <g transform="scale(1,2)">
    <path d="M 50,50 C 400,100 200,200 100,20"
      fill="blue" stroke="black" stroke-width="4"/>
  </g>
</g>

<!-- translate/scale a blue cubic Bezier curve -->
<g transform="translate(-50,50)">
  <g transform="scale(0.5, 0.5) translate(195,345)">
    <path d="m20,20 C20,50 20,450 300,200 s-150,-250 200,100"
      fill="blue" style="stroke:#880088;stroke-width:4;"/>
  </g>

  <!-- scale a cubic Bezier curve with 'pattern2' -->
  <g transform="scale(0.5, 0.5) translate(185,335)">
    <path d="m20,20 C20,50 20,450 300,200 s-150,-250 200,100"
      fill="url(#pattern2)"
      style="stroke:#880088;stroke-width:4;"/>
  </g>

  <!-- scale a reddish blue cubic Bezier curve -->
  <g transform="scale(0.5, 0.5) translate(180,330)">
    <path d="m20,20 C20,50 20,450 300,200 s-150,-250 200,100"
      fill="blue" style="stroke:#880088;stroke-width:4;"/>
  </g>

  <g transform="scale(0.5, 0.5) translate(170,320)">
    <path d="m20,20 C20,50 20,450 300,200 s-150,-250 200,100"
      fill="url(#pattern2)"
      style="stroke:black;stroke-width:4;"/>
  </g>
</g>

<g transform="scale(0.8,1) translate(380,120)">
  <path d="M0,0 C200,150 400,300 20,250"
    fill="url(#pattern2)"
    style="stroke:blue;stroke-width:4;"/>
</g>

<g transform="scale(2.0,2.5) translate(150,-80)">
  <path d="M200,150 C0,0 400,300 20,250"
    fill="url(#pattern2)"
    style="stroke:blue;stroke-width:4;"/>
</g>
</svg>

```

Listing 5.4 contains an SVG `<defs>` element that defines two linear gradients followed by ten SVG `<path>` elements, each of which renders a cubic

Bezier curve. The SVG `<path>` elements are enclosed in SVG `<g>` elements whose transform attribute contains the SVG `scale()` function or the SVG `translate()` function (or both).

The first SVG `<g>` element invokes the SVG `scale()` function to scale the cubic Bezier curve that is specified in an SVG `<path>` element, as shown here:

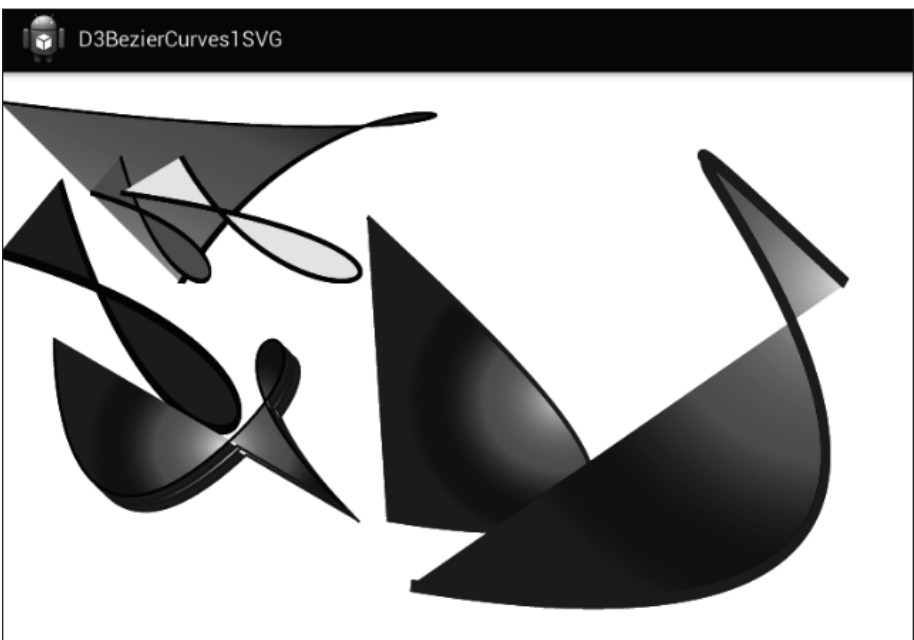
```
<g transform="scale(1.5,0.5)">
  <path d="m 0,50 C 400,200 200,-150 100,350"
        stroke="black" stroke-width="4"
        fill="url(#pattern1)" />
</g>
```

The preceding cubic Bezier curve has an initial point  $(0, 50)$  with control points  $(400, 200)$  and  $(200, -150)$  followed by the second control point  $(100, 350)$ . The Bezier curve is black, with a width of 4, and its fill attribute is the color defined in the `<linearGradient>` element (whose `id` attribute has value `pattern1`) that is defined in the SVG `<defs>` element.

The remaining SVG `<path>` elements are similar to the first SVG `<path>` element, so they will not be described.

The code for quadratic and Bezier curves uses a `<path>` element in SVG; in an earlier example you saw how to write D3 code that uses a path, so we won't repeat that code here.

Figure 5.4 displays the result of rendering the Bezier curves that are defined in the SVG document `BezierCurves1.svg` in a landscape-mode



**FIGURE 5.4** SVG Bezier Curves on an Asus Prime Android 4.0.3 10-inch Tablet.

screenshot taken from an Android application running on an Asus Prime Android 4.0.3 10-inch tablet.

## SVG FILTERS AND SHADOW EFFECTS

You can create nice filter effects that you can apply to 2D shapes as well as text strings, and this section contains three SVG-based examples of creating such effects. Listing 5.5 displays the contents of the SVG documents `BlurFilterText1.svg`.

### LISTING 5.5 *BlurFilterText1.svg*

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20001102//EN"
"http://www.w3.org/TR/2000/CR-SVG-20001102/DTD/svg-20001102.dtd">

<svg xmlns="http://www.w3.org/2000/svg"
      xmlns:xlink="http://www.w3.org/1999/xlink"
      width="100%" height="100%">
  <defs>
    <filter
      id="blurFilter1"
      filterUnits="objectBoundingBox"
      x="0" y="0"
      width="100%" height="100%">
      <feGaussianBlur stdDeviation="4"/>
    </filter>
  </defs>

  <g transform="translate(50,100)">
    <text id="normalText" x="0" y="0"
          fill="red" stroke="black" stroke-width="4"
          font-size="72">
      Normal Text
    </text>

    <text id="horizontalText" x="0" y="100"
          filter="url(#blurFilter1)"
          fill="red" stroke="black" stroke-width="4"
          font-size="72">
      Blurred Text
    </text>
  </g>
</svg>
```

The SVG `<defs>` element in Listing 5.5 contains an SVG `<filter>` element that specifies a Gaussian blur with the following line:

```
<feGaussianBlur stdDeviation="4"/>
```

You can specify larger values for the `stdDeviation` attribute if you want to create more diffuse filter effects.



**FIGURE 5.5** SVG Filter Effect.

The first SVG `<text>` element that is contained in the SVG `<g>` element renders a normal text string, whereas the second SVG `<text>` element contains a `filter` attribute that references the filter (defined in the SVG `<defs>` element) to render the same text string, as shown here:

```
filter="url(#blurFilter1)"
```

Figure 5.5 displays the result of rendering `BlurFilterText1.svg` that creates a filter effect in Google Chrome on a Macbook (SVG filters are not supported in Android 4.0.3).

## RENDERING TEXT ALONG AN SVG `<PATH>` ELEMENT

SVG enables you to render a text string along an SVG `<path>` element. Listing 5.6 displays the contents of the document `TextOnQBezierPath1.svg` that illustrate how to render a text string along the path of a quadratic Bezier curve.

### **LISTING 5.6** *TextOnQBezierPath1.svg*

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20001102//EN"
  "http://www.w3.org/TR/2000/CR-SVG-20001102/DTD/svg-20001102.dtd">

<svg xmlns="http://www.w3.org/2000/svg"
      xmlns:xlink="http://www.w3.org/1999/xlink"
      width="100%" height="100%">
  <defs>
    <path id="pathDefinition"
          d="m0,0 Q100,0 200,200 T300,200 z"/>
  </defs>
```

```

<g transform="translate(100,100)">
  <text id="textStyle" fill="red"
        stroke="blue" stroke-width="2"
        font-size="24">

    <textPath xlink:href="#pathDefinition">
      Sample Text that follows a path specified by a Quadratic Bezier curve
    </textPath>
  </text>
</g>
</svg>

```

The SVG `<defs>` element in Listing 5.6 contains an SVG `<path>` element that defines a quadratic Bezier curve (note the `Q` in the `d` attribute). This SVG `<path>` element has an `id` attribute whose value is `pathDefinition`, which is referenced later in this code sample.

The SVG `<g>` element contains an SVG `<text>` element that specifies a text string to render, as well as an SVG `<textPath>` element that specifies the path along which the text is rendered, as shown here:

```

<textPath xlink:href="#pathDefinition">
  Sample Text that follows a path specified by a Quadratic Bezier curve
</textPath>

```

Notice that the SVG `<textPath>` element contains the attribute `xlink:href`, whose value is `pathDefinition`, which is also the `id` of the SVG `<path>` element that is defined in the SVG `<defs>` element. As a result, the text string is rendered along the path of a quadratic Bezier curve instead of rendering the text string horizontally (which is the default behavior).

Figure 5.6 displays the result of rendering `TextOnQBezierPath1.svg` that renders a text string along the path of a quadratic Bezier curve in a landscape-mode screenshot taken from an Android application running in a Chrome Browser on a Macbook Pro.

## SVG TRANSFORMS

Earlier in this chapter you saw some examples of SVG transform effects. In addition to the SVG functions `scale()`, `translate()`, and `rotate()`, SVG provides the `skew()` function to create skew effects.

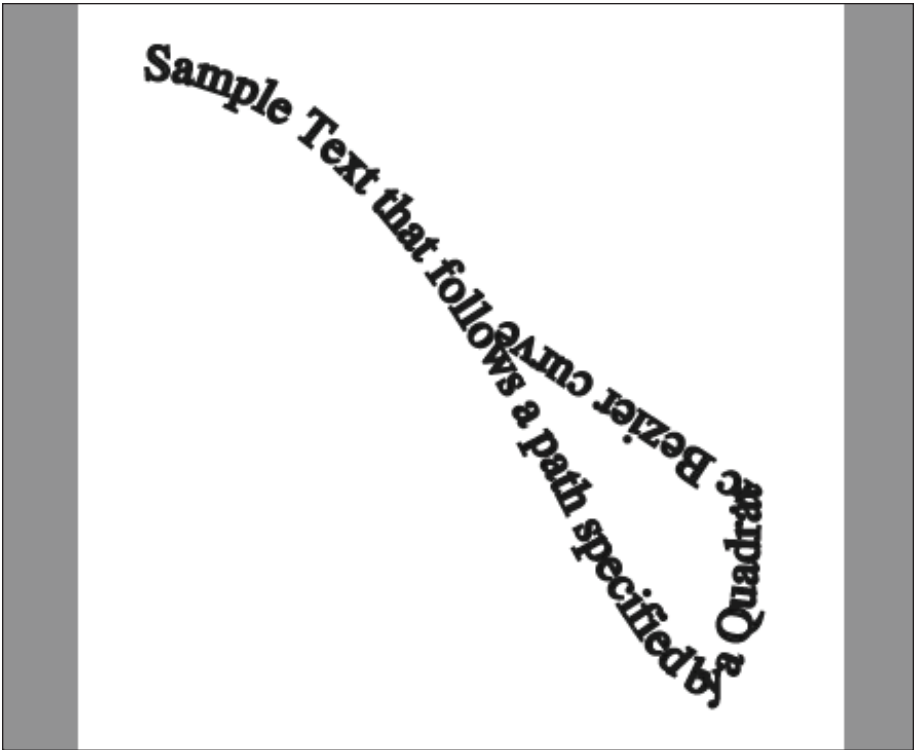
Listing 5.7 displays the contents of `TransformEffects1.svg` that illustrates how to apply transforms to rectangles and circles in SVG.

### **LISTING 5.7** *TransformEffects1.svg*

```

<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20001102//EN"
  "http://www.w3.org/TR/2000/CR-SVG-20001102/DTD/svg-20001102.dtd">

```



**FIGURE 5.6** SVG Text on a Quadratic Bezier in a Chrome Browser on a Macbook Pro.

```
<svg xmlns="http://www.w3.org/2000/svg"
    xmlns:xlink="http://www.w3.org/1999/xlink"
    width="100%" height="100%">
<defs>
  <linearGradient id="gradientDefinition1"
    x1="0" y1="0" x2="200" y2="0"
    gradientUnits="userSpaceOnUse">
    <stop offset="0%" style="stop-color:#FF0000"/>
    <stop offset="100%" style="stop-color:#440000"/>
  </linearGradient>

  <pattern id="dotPattern" width="8" height="8"
    patternUnits="userSpaceOnUse">

    <circle id="circle1" cx="2" cy="2" r="2"
      style="fill:red;"/>
  </pattern>
</defs>

<!-- full cylinder -->
<g id="largeCylinder" transform="translate(100,20)">
  <ellipse cx="0" cy="50" rx="20" ry="50"
    stroke="blue" stroke-width="4"
    style="fill:url(#gradientDefinition1)"/>
```



```

<rect x="0" y="0" width="300" height="100"
      style="fill:url(#gradientDefinition1)"/>

<rect x="0" y="0" width="300" height="100"
      style="fill:url(#dotPattern)"/>

<ellipse cx="300" cy="50" rx="20" ry="50"
         stroke="blue" stroke-width="4"
         style="fill:yellow;"/>
</g>

<!-- half-sized cylinder -->
<g transform="translate(100,100) scale(.5)">
  <use xlink:href="#largeCylinder" x="0" y="0"/>
</g>

<!-- skewed cylinder -->
<g transform="translate(100,100) skewX(40) skewY(20)">
  <use xlink:href="#largeCylinder" x="0" y="0"/>
</g>

<!-- rotated cylinder -->
<g transform="translate(100,100) rotate(40)">
  <use xlink:href="#largeCylinder" x="0" y="0"/>
</g>
</svg>

```

The SVG `<defs>` element in Listing 5.7 contains a `<linearGradient>` element that defines a linear gradient followed by an SVG `<pattern>` element that defines a custom pattern, which is shown here:

```

<pattern id="dotPattern" width="8" height="8"
         patternUnits="userSpaceOnUse">

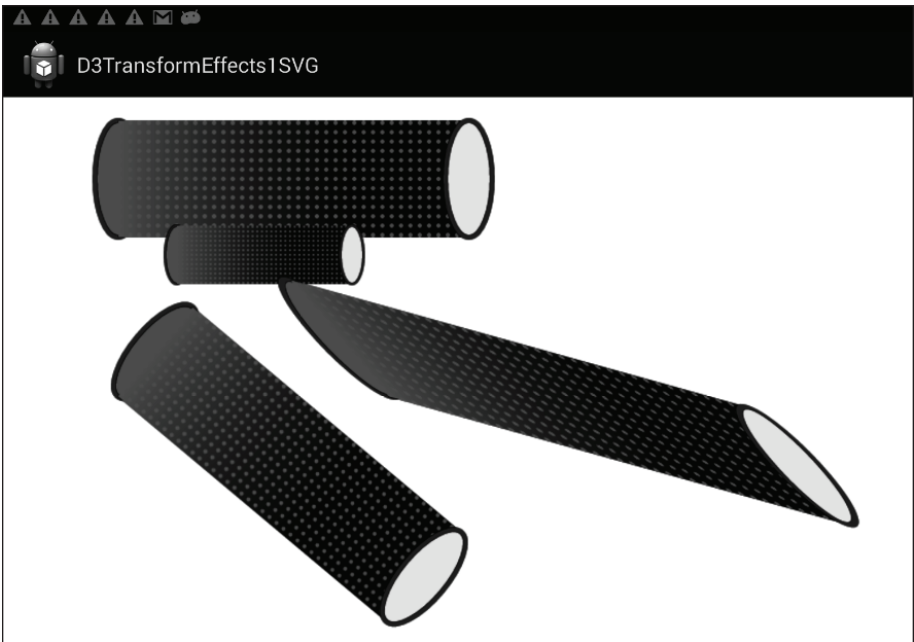
  <circle id="circle1" cx="2" cy="2" r="2"
         style="fill:red;"/>
</pattern>

```

As you can see, the SVG `<pattern>` element contains an SVG `<circle>` element that is repeated in a grid-like fashion inside an 8x8 pixel rectangle (note the values of the `width` attribute and the `height` attribute). The SVG `<pattern>` element has an `id` attribute whose value is `dotPattern` because (as you will see) this element creates a dotted effect.

Listing 5.7 contains four SVG `<g>` elements, each of which renders a cylinder that references the SVG `<pattern>` element that is defined in the SVG `<defs>` element.

The first SVG `<g>` element in Listing 5.7 contains two SVG `<ellipse>` elements and two SVG `<rect>` elements. The first `<ellipse>` element renders the left-side cover of the cylinder with the linear gradient that is defined in the SVG `<defs>` element. The first `<rect>` element renders the body of the cylinder with a linear gradient, and the second `<rect>` element



**FIGURE 5.7** SVG Transform Effects on an Asus Prime Android 4.0.3 10-inch Tablet.

renders the dot pattern on the body of the cylinder. Finally, the second `<ellipse>` element renders the right-side cover of the ellipse.

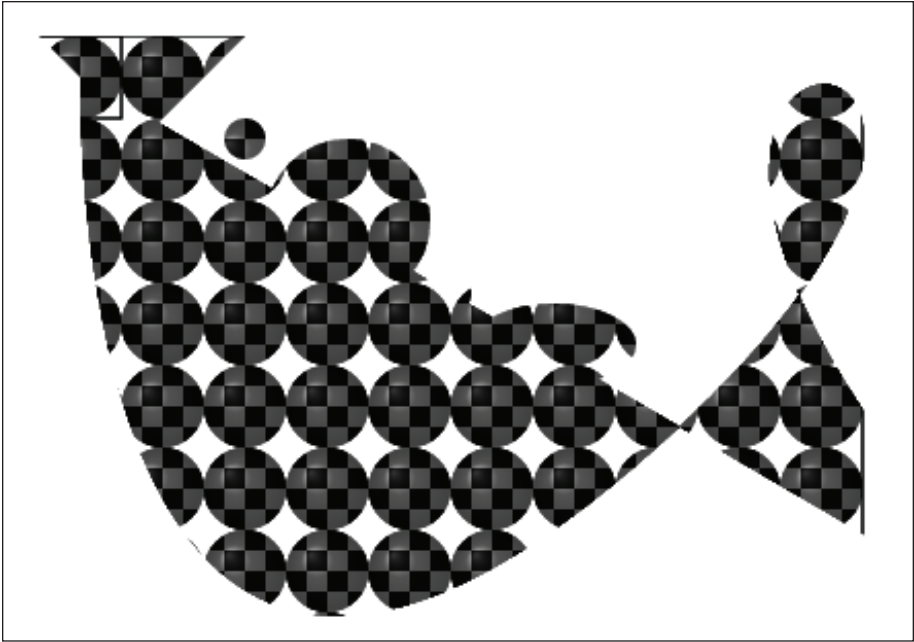
The other three cylinders are easy to create: they simply reference the first cylinder and apply a transformation to change the size, shape, and orientation. Specifically, these three cylinders reference the first cylinder with the code `<use xlink:href="#largeCylinder" x="0" y="0"/>` and then they apply scale, skew, and rotate functions to render scaled, skewed, and rotated cylinders.

Figure 5.7 displays the result of rendering `TransformEffects1.svg`, in a landscape-mode screenshot taken from an Android application running on an Asus Prime Android 4.0.3 10-inch tablet.

## THE SVG `<CLIPPATH>` ELEMENT

The SVG `<clipPath>` element enables you to clip a rendered image. This element allows you to embed many other SVG elements including complex SVG `<path>` definitions. The SVG document `DrcBezier3D-CircleCB13.svg` (about two pages of code) and the HTML Web page `DrcBezier3DCircleCB13.html` (almost four pages of code) are on the accompanying CD.





**FIGURE 5.8** SVG <clipPath> in a Chrome Browser on a Macbook Pro.

The syntax of the SVG <clipPath> element is straightforward, as shown here:

```
<clipPath id="clipPathDefinition"
  clipPathUnits="userSpaceOnUse">
  <path d="m20,20 C20,50 20,450 300,200 s-150,-250 200,100"
    fill="blue"
    stroke-dasharray="2 2 2 2"
    style="stroke:#880088;stroke-width:4;"/>
  // other elements omitted for brevity
</clipPath>
```

Figure 5.8 displays the result of rendering `DrcBezier3DCircleCB13.svg` in a Chrome browser on a Macbook Pro.

## OTHER SVG FEATURES

SVG supports other features (such as animation), and you can also combine SVG with CSS selectors (discussed in Chapter 6). If needed, you can embed JavaScript code inside a CDATA (character data) section in an SVG document. Although SVG provides support only for 2D shapes, you can use JavaScript to create 3D effects in SVG. Moreover, because SVG documents are XML documents, you can also apply XSL (Extensible Stylesheet Language) stylesheets to SVG documents. The next several sections provide more information about



these SVG features; keep in mind that the referenced files are located on the CD that accompanies this book.

## SVG Animation

SVG supports animation effects that you can specify as part of the declaration of SVG elements. The SVG document `AnimateMultiRect1.svg` illustrates how to create an animation effect with four rectangles.

```
<rect id="rect1" width="100" height="100"
      stroke-width="1" stroke="blue"/>

<use xlink:href="#rect1" x="0" y="0" fill="red">
  <animate attributeName="x" attributeType="XML"
    begin="0s" dur="4s"
    fill="freeze" from="0" to="400"/>
</use>
```

The SVG `<g>` element contains an SVG `<use>` element that performs a parallel animation effect on a rectangle. The first `<use>` element references the rectangle defined in the SVG `<defs>` element and then animates the `x` attribute during a four-second interval. Notice that the `x` attribute varies from 0 to 400, which moves the rectangle horizontally from left to right. You can also animate other SVG elements including text strings.

If you are interested in seeing how to handle zoom and pan functionality in SVG, code samples are available at [http://msdn.microsoft.com/en-us/library/gg589508\(v=vs.85\).aspx](http://msdn.microsoft.com/en-us/library/gg589508(v=vs.85).aspx).

## Creating Three-Dimensional Effects in SVG

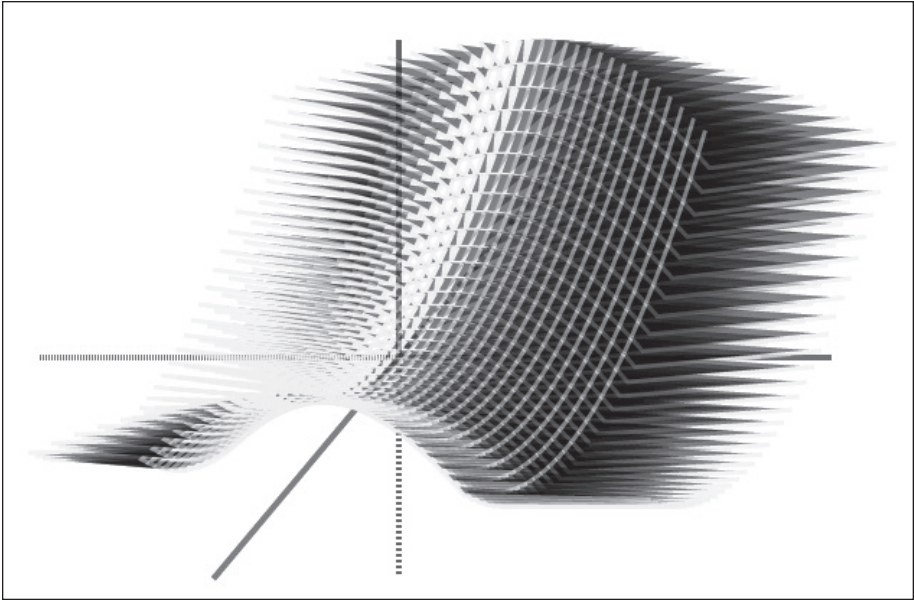
Although SVG does not provide 3D support, you can create 3D-like effects based on a combination of JavaScript and SVG. An open-source project with more than one thousand SVG code samples that illustrate how to create 3D effects can be found at <http://code.google.com/p/svg-filter-graphics>.



The CD contains the HTML Web page `TroughPattern3S2.html` that is the counterpart of `TroughPattern3S2.svg` (which is part of the preceding open-source project). The CD also contains several augmented versions of this same code sample that create animation effects. Keep in mind that most versions of Android do not support SVG filters, so the trough-related code samples do not contain an SVG `<filter>` element.

Figure 5.9 displays the result of rendering `TroughPattern3S2.html` in a Chrome browser on a Macbook Pro.

If you are interested in creating 3D effects with SVG, you can get more information (including details about matrix manipulation) and code samples here at [http://msdn.microsoft.com/en-us/library/hh535759\(v=vs.85\).aspx](http://msdn.microsoft.com/en-us/library/hh535759(v=vs.85).aspx). The toolkits D3 and Raphael generate SVG code, and many code samples are found at <http://code.google.com/p/d3-graphics/> and <http://code.google.com/p/raphael-graphics/>.



**FIGURE 5.9** SVG 3D Trough Shape in a Chrome Browser on a Macbook.

## SVG and HTML

You can use an `<svg>` element in an HTML5 Web page to embed pure SVG code. For example, Listing 5.8 renders a red rectangle with a blue border.

### **LISTING 5.8** *EmbeddedSVG.html*

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="utf-8" />
  <title>Embedded SVG</title>
</head>

<body>
<svg>
  <rect x="20" y="20" width="200" height="100"
    fill="red" stroke="blue" stroke-width="4"/>
</svg>
</body>
</html>
```

The code in Listing 5.8 is straightforward: it consists of HTML5 boilerplate text and an SVG `<rect>` element that you have seen earlier in this chapter.

## SVG and JavaScript

In addition to embedding pure SVG code in an HTML5 page, SVG allows you to embed JavaScript in a CDATA section to dynamically create SVG

elements and append them to the DOM of the HTML Web page. You can also define event listeners in JavaScript for SVG elements. The SVG document `ArchEllipses1.svg` renders a set of ellipses that follow the path of an Archimedean spiral. A fragment is shown here:

```
var svgNS = "http://www.w3.org/2000/svg";

function drawSpiral(event) {
    for(angle=0; angle<maxAngle; angle+=angleDelta) {
        ellipseNode = svgDocument.createElementNS(svgNS, "ellipse");
        ellipseNode.setAttribute("fill", redColor);
        ellipseNode.setAttribute("stroke-width", strokeWidth);
        ellipseNode.setAttribute("stroke", "yellow");

        ellipseNode.setAttribute("cx", currentX);
        ellipseNode.setAttribute("cy", currentY);
        ellipseNode.setAttribute("rx", majorAxis);
        ellipseNode.setAttribute("ry", minorAxis);
        gcNode.appendChild(ellipseNode);
    }
}
```

The SVG document `ArchEllipses1.svg` contains a CDATA section with a `<script>` element, which in turn contains the `drawSpiral()` function whose main loop renders a set of dynamically created SVG `<ellipse>` elements. Each SVG `<ellipse>` element is created in the SVG namespace that is specified in the variable `svgNS`, after which values are assigned to the required attributes of an ellipse. After each SVG `<ellipse>` element is dynamically created, the element is appended to the DOM.

## CSS3 AND SVG

CSS3 selectors can reference SVG documents using the `CSS3 url()` function, which means that you can incorporate SVG-based graphics effects (including animation) in your HTML pages. For example, the following code block references the SVG document `Blue3DCircle1.svg` in a CSS selector:

```
#circle1 {
    opacity: 0.5; color: red;
    width: 250px; height: 250px;
    position: absolute; top: 0px; left: 0px;
    font-size: 24px;
    -webkit-border-radius: 4px;
    -moz-border-radius: 4px;
    border-radius: 4px;
    -webkit-background: url(Blue3DCircle1.svg) top right;
    -moz-background: url(Blue3DCircle1.svg) top right;
    background: url(Blue3DCircle1.svg) top right;
}
```

## CSS3 and SVG Bar Charts



Due to space constraints, this topic is not covered in this chapter, but the CD contains the HTML5 Web page `CSS3SVGBarLayout1.html` that references both the CSS stylesheet `CSS3SVGBarLayout1.css` and the SVG document `CSS3SVGBarLayout1.svg`. This Web page renders an SVG circle, bar chart, and multicolumn text.

## SIMILARITIES AND DIFFERENCES BETWEEN SVG AND CSS3

This section briefly summarizes the features that are common to SVG and CSS3 as well as the features that are unique to each technology. Chapter 6 contains code samples that illustrate many of the features that are summarized in this section.

SVG and CSS3 both provide support for the following:

- linear and radial gradients
- 2D graphics and animation effects
- shapes such as rectangles, circles, and ellipses
- WAI ARIA (Web Accessibility Initiative - Accessible Rich Internet Applications)

SVG provides support for the following features that are not available in CSS3:

- Bezier curves
- hierarchical object definitions
- custom glyphs
- rendered text along an arbitrary path
- defined event listeners on SVG objects
- programmatic creation of 2D shapes using JavaScript
- accessibility to XML-based technologies and tools

CSS3 provides support for the following features that are not available in SVG:

- 3D graphics and animation effects
- multicolumn rendering of text
- WebGL-oriented functionality (e.g., CSS shaders)

Note that SVG filters and CSS filters will become one and the same at some point in the not-too-distant future.

In general, SVG is better suited than CSS3 for large data sets that will be used for data visualization, and you can reference the SVG document (which might render some type of chart) in a CSS3 selector using the `CSS3 url()` function. You have already seen such an example in chapter three, where the SVG document contains the layout for a bar chart. In general, there might be additional processing involved where data is retrieved or aggregated from one or more sources (such as databases and Web services) and then manipulated using some programming language (such as XSLT (Extensible Stylesheet Language Transformations), Java, or JavaScript) to programmatically create an SVG document or perhaps create SVG elements in a browser session.

## SVG AND XSLT (EXTENSIBLE STYLESHEET LANGUAGE TRANSFORMATIONS)

XSLT (XML StyleSheet Transforms) is a technology involving XSL stylesheets. In many cases, an XSL stylesheet is applied to an XML-based input document to generate XML documents (as well as other file formats) as output. Typically the XSL stylesheet (perhaps written by you) contains so-called templates that are invoked when they match elements in an input XML document. After the processing is completed, the newly created output document can be used as a final document or as the input document for another XSL stylesheet (so you can perform a pipeline style of processing). In particular, you can apply an XSL stylesheet to an XML input document to generate an SVG document.



The CD contains the XSL stylesheet `Simple.xsl` that is applied to the XML document `Simple.xml` to generate an SVG document. The purpose of these two files is to give you an idea of the sort of code that you will work with if you decide to use XSLT. Although XSLT is a complex topic (and beyond the scope of this book), there are many online tutorials about XSLT that provide useful information.

### ADDITIONAL CODE SAMPLES ON THE CD



As you have seen in earlier chapters, the supplemental code samples provide you with tips and techniques for creating various visual effects. You can selectively choose whichever effects you deem useful for your custom code.

There are several HTML Web pages that render regular polygons and star patterns along with Gaussian blur effects and shadow backgrounds using JavaScript embedded in an SVG document. These samples include:

- `Polygons1.svg`
- `Polygons1BlurFilter1.svg`
- `Stars1.svg`
- `Stars1BlurFilter1.svg`
- `Stars1Shadow1.svg`

The HTML Web page `FollowTheMouse2BlurFilter1.html` is the counterpart to the HTML Web page `FollowTheMouse2.html` augmented by a Gaussian blur filter. Unlike the two preceding examples, the SVG documents `Polygons1.svg` and `Polygons1BlurFilter1.svg` use embedded JavaScript to render a set of polygons. The latter code sample also applies a Gaussian blur effect to the polygons.

The SVG document `LineSegments1.svg` shows you a technique for creating linear gradients in JavaScript without using the built-in SVG support for linear gradients. Compare the JavaScript code in the SVG document `LineSegments1.svg` with the corresponding D3 code in the HTML Web page `LineSegments1.html`. A mouseover event handler in D3 is



specified in three lines of code for each line segment. The first block of line segments specifies a `scale` transform; the second and third blocks specify a `rotate` transform and a `skewX` transform, respectively. You need to move your mouse slowly because it's easy to inadvertently skip over line segments, but you can always retrace your movements so that every line segment has its transform applied to it.

## SUMMARY

---

This chapter started with examples of combining SVG with CSS3 followed by an overview of some of the exciting new features of CSS3. In particular, you learned how to create the following shapes in SVG:

- line segments
- rectangles
- circles, ellipses, and arcs
- Bezier curves

The next chapter introduces you to CSS3 and shows you how to create rich graphics and animation effects with CSS3.

# *INTRODUCTION TO CSS3 GRAPHICS AND ANIMATION*

**T**his chapter introduces various aspects of CSS3 such as 2D/3D graphics and 2D/3D animation. By necessity, you need a basic understanding of CSS3 for this chapter (such as how to set properties in CSS selectors). If you are unfamiliar with CSS selectors, there are many introductory articles available through an Internet search. If you are convinced that CSS operates under confusing and seemingly arcane rules, then it's probably worth your while to read an online article about CSS box rules, after which you will have a better understanding of the underlying logic of CSS. In some cases, CSS3 concepts are presented without code samples due to space limitations; however, those concepts are included because it's important for you to be aware of their existence.

The first part of this chapter contains code samples that illustrate how to create shadow effects, how to render rectangles with rounded corners, and how to use linear and radial gradients. The second part of this chapter covers CSS3 transforms (scale, rotate, skew, and translate), along with code samples that illustrate how to apply transforms to HTML elements and JPG files. The third part of this chapter covers CSS3 3D graphics and animation effects, and the fourth part of this chapter briefly discusses CSS3 Media Queries, which enable you to detect some characteristics of a device, and therefore render an HTML5 Web page based on those properties. The final portion of this chapter shows you how to reference SVG documents in CSS selectors via the `url()` function. This is very powerful functionality because it enables you to leverage the power of SVG in CSS3.

You can launch the code samples in this chapter in a Webkit-based browser (see comments in the Preface) on a desktop or a laptop; you can also view them on mobile devices, provided that you launch them in a browser that supports the CSS3 features that are used in the code samples. For your convenience,

some screenshots of the code samples in this chapter were taken on mobile devices including a Nexus 7 2 with Android JellyBean 4.3 and an iPad3.

After reading this chapter you will be able to understand the underlying code in many of the CSS3 samples that you can create on Adobe's CSS Filter-Lab: <http://www.adobe.com/devnet/html5/articles/css-filterlab.html>.

In addition, an open-source project with more than one thousand swatch-like code samples for graphics and animation effects using pure CSS3 (no JavaScript, Canvas, or SVG) and is here: <https://github.com/ocampesato/css3-graphics>.

## CSS3 SUPPORT AND BROWSER-SPECIFIC PREFIXES FOR CSS3

Before we delve into the details of CSS3, there are two important details that you need to know about defining CSS3-based selectors for HTML pages. First, you need to know the CSS3 features that are available in different browsers. One of the best Websites for determining browser support for CSS3 features is <http://caniuse.com/>. This link contains tabular information regarding CSS3 support in Internet Explorer, Firefox, Safari, Chrome, and Opera.

Another highly useful tool that checks for CSS3 feature support is `Enhance.js` that tests browsers to determine whether they can support a set of essential CSS and JavaScript properties and delivers features to those browsers that satisfies the test. You can download `Enhance.js` at [http://filamentgroup.com/lab/introducing\\_enhancejs\\_smarter\\_safer\\_apply\\_progressive\\_enhancement/](http://filamentgroup.com/lab/introducing_enhancejs_smarter_safer_apply_progressive_enhancement/).

The second detail that you need to know is that many CSS3 properties currently require browser-specific prefixes to work correctly. The prefixes `-ie-`, `-moz-`, and `-o-` are for Internet Explorer, Firefox, and Opera, respectively. As an illustration, the following code block shows examples of these prefixes:

```
-ie-webkit-border-radius: 8px;
-moz-webkit-border-radius: 8px;
-o-webkit-border-radius: 8px;
border-radius: 8px;
```

In your CSS selectors, specify the attributes with browser-specific prefixes before the generic attribute, which serves as a default choice in the event that the browser-specific attributes are not selected. The CSS3 code samples in this book contain `WebKit`-specific prefixes, which helps us keep the CSS stylesheets manageable in terms of size. If you need CSS stylesheets that work on multiple browsers (for current versions as well as older versions), there are essentially two options available. One option involves manually adding the CSS3 code with all the required browser-specific prefixes, which can be tedious to maintain and also error-prone. Another option is to use CSS toolkits or frameworks (discussed in the next chapter) that can programmatically generate the CSS3 code that contains all browser-specific prefixes. Finally, an extensive list of browser-prefixed CSS properties can be found at <http://peter.sh/experiments/vendor-prefixed-css-property-overview/>.

## QUICK OVERVIEW OF CSS3 FEATURES

---

CSS3 adopts a modularized approach for extending existing CSS2 functionality as well as supporting new functionality. As such, CSS3 can logically be divided into the following categories:

- backgrounds/borders
- color
- media queries
- multicolumn layout
- selectors

With CSS3 you can create boxes with rounded corners and shadow effects, create rich graphics effects using linear and radial gradients, detect portrait and landscape mode, detect the type of mobile device using media-query selectors, and produce multicolumn text rendering and formatting.

In addition, CSS3 enables you to define sophisticated node-selection rules in selectors using pseudo-classes, first or last child (`first-child`, `last-child`, `first-of-type`, and `last-of-type`), and also pattern-matching tests for attributes of elements. Several sections in this chapter contain examples of how to create such selection rules.

## CSS3 PSEUDOCASSES, ATTRIBUTE SELECTION, AND RELATIONAL SYMBOLS

---

This brief section contains examples of some pseudoclasses followed by snippets that show you how to select elements based on the relative position of text strings in various attributes of those elements.

Recall that the `class` attribute in CSS provides a way to mark a group of elements as having a certain property, such as the following code snippet:

```
<p class="details">
```

On the other hand, pseudoclasses enable you to reference an arbitrary group of elements by some identifying feature that they possess. For example, the following code snippet collects the first paragraph in each HTML `<section>` element in an HTML Web page:

```
section:p:first-of-type
```

CSS3 supports an extensive and rich set of pseudoclasses, including `nth-child()`, along with some of its semantically related variants such as `nth-of-type()`, `nth-first-of-type()`, `nth-last-of-type()`, and `nth-last-child()`.

CSS3 also supports Boolean selectors (which are also pseudoclasses) such as `empty`, `enabled`, `disabled`, and `checked`, which are very useful

for Form-related HTML elements. One other pseudoclass is `not()`, which returns a set of elements that do not match the selection criteria.

CSS3 uses the metacharacters `^`, `$`, and `*` (followed by the `=` symbol) to match an initial, terminal, or arbitrary position for a text string. If you are familiar with the Unix utilities `grep` and `sed`, as well as the `vi` text editor, then these metacharacters are very familiar to you. However, CSS3 imposes a restriction for using metacharacters: they can only be used in the context of an attribute match (which uses square brackets).

## CSS3 Pseudoclasses

The CSS3 `nth-child()` is a very powerful and useful pseudoclass, and it has the following form:

```
nth-child(insert-a-keyword-or-linear-expression-here)
```

The following list provides various examples of using the `nth-child()` pseudoclass to match various subsets of child elements of an HTML `<div>` element (which can be substituted by other HTML elements as well):

- `div:nth-child(1)`: matches the first child element
- `div:nth-child(2)`: matches the second child element
- `div:nth-child(even)`: matches the even child elements
- `div:nth-child(odd)`: matches the odd child elements

The interesting and powerful aspect of the `nth-child()` pseudoclass is its support for linear expressions of the form `an+b`, where `a` is a positive integer, and `b` is a nonnegative integer, as shown here (using an HTML5 `<div>` element):

```
div:nth-child(3n): matches every third child, starting from position 0
div:nth-child(3n+1): matches every third child, starting from position 1
div:nth-child(3n+2): matches every third child, starting from position 2
```

## CSS3 Attribute Selection

You can specify CSS3 selectors that select HTML elements as well as HTML elements based on the value of an attribute of an HTML element using various regular expressions. For example, the following selector selects `img` elements whose `src` attribute starts with the text string `Fiona` and then sets the `width` attribute and the `height` attribute of the selected `img` elements to `100px`:

```
img[src^="Fiona"] {
    width: 100px; height: 100px;
}
```

The preceding CSS3 selector is useful when you want to set different dimensions to images based on the name of the images (`Fiona`, `Marci`, `Sarah`, and so forth).

The following HTML `<img>` elements do not match the preceding selector:

```


```

The following selector selects HTML `img` elements whose `src` attribute ends with the text string `jpeg` and then sets the `width` attribute and the `height` attribute of the selected `img` elements to 150px:

```
img[src$="jpeg"] {
    width: 150px; height: 150px;
}
```

The preceding CSS3 selector is useful when you want to set different dimensions to images based on the type of the images (`jpg`, `png`, `jpeg`, and so forth).

The following selector selects HTML `img` elements whose `src` attribute contains any occurrence of the text string `baby` and then sets the `width` attribute and the `height` attribute of the selected HTML `img` elements to 200px:

```
img[src*="baby"] {
    width: 200px; height: 200px;
}
```

The preceding CSS3 selector, which uses the `*` metacharacter, is useful when you want to set different dimensions to images based on the classification of the images (`mybaby`, `yourbaby`, `babygirl`, `babyboy`, and so forth).

If you want to learn more about patterns (and their descriptions) that you can use in CSS3 selectors, an extensive list is available at <http://www.w3.org/TR/css3-selectors>.

## CSS3 SHADOW EFFECTS AND ROUNDED CORNERS

---

CSS3 shadow effects are useful for creating vivid visual effects with simple selectors. You can use shadow effects for text as well as rectangular regions. CSS3 also enables you to easily render rectangles with rounded corners, so you do not need JPG files to create this effect.

### Specifying Colors with Red/Green/Blue Triples and Hue/Saturation/Lightness Representations

---

Before we delve into the interesting features of CSS3, you need to know how to represent colors. One method is to use triples which represent the Red, Green, and Blue (RGB) components of a color. For instance, the triples `(255, 0, 0)`, `(255, 255, 0)`, and `(0, 0, 255)` represent the colors Red, Yellow, and Blue. Other ways of specifying the color include: the hexadecimal triples `(FF, 0, 0)` and `(FF, 0, 0)`, the decimal triple `(100%, 0, 0)`, or the string `#F00`. You can also use `(R, G, B, A)`, where the fourth component

(often called the alpha value) specifies the opacity, which is a decimal number between 0 (invisible) to 1 (opaque), inclusive. However, there is also the Hue, Saturation, and Luminosity (HSL) representation of colors, where the first component is an angle between 0 and 360 (0 degrees is north), and the other two components are percentages between 0 and 100. For instance, the three triples (0, 100%, 50%), (120, 100%, 50%), and (240, 100%, 50%) represent the colors Red, Green, and Blue, respectively.

Although the code samples in this book use (R, G, B) and (R, G, B, A) for representing colors, Wikipedia contains a very good HSL entry that you ought to read if you want to learn more about HSL.

## CSS3 and Text Shadow Effects

A shadow effect for text can make a Web page look more vivid and appealing, and many Websites look better with shadow effects that are not overpowering for users (unless you specifically need to do so).

Listing 6.1 displays the contents of the HTML5 page `TextShadow1.html` that illustrate how to render text with a shadow effect, and Listing 6.2 displays the contents of the CSS stylesheet `TextShadow1.css` that is referenced in Listing 6.1.

### LISTING 6.1 *TextShadow1.html*

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="utf-8" />
  <title>CSS Text Shadow Example</title>
  <link href="TextShadow1.css" rel="stylesheet" type="text/css">
</head>

<body>
  <div id="text1">Line One Shadow Effect</div>
  <div id="text2">Line Two Shadow Effect</div>
  <div id="text3">Line Three Vivid Effect</div>
  <div id="text4">
    <span id="dd">13</span>
    <span id="mm">August</span>
    <span id="yy">2012</span>
  </div>
  <div id="text5">
    <span id="dd">13</span>
    <span id="mm">August</span>
    <span id="yy">2012</span>
  </div>
  <div id="text6">
    <span id="dd">13</span>
    <span id="mm">August</span>
    <span id="yy">2012</span>
  </div>
</body>
</html>
```

The code in Listing 6.1 is straightforward: there is a reference to the CSS stylesheet `TextShadow1.css` that contains two CSS selectors. One selector specifies how to render the HTML `<div>` element whose `id` attribute has value `text1`, and the other selector matches the HTML `<div>` element whose `id` attribute is `text2`. Although the CSS3 `rotate()` function is included in this example, we'll defer a more detailed discussion of this function until later in this chapter.

#### **LISTING 6.2** *TextShadow1.css*

```
#text1 {
    font-size: 24pt;
    text-shadow: 2px 4px 5px #00f;
}

#text2 {
    font-size: 32pt;
    text-shadow: 0px 1px 6px #000,
                4px 5px 6px #f00;
}

#text3 {
    font-size: 40pt;
    text-shadow: 0px 1px 6px #fff,
                2px 4px 4px #0ff,
                4px 5px 6px #00f,
                0px 0px 10px #444,
                0px 0px 20px #844,
                0px 0px 30px #a44,
                0px 0px 40px #f44;
}

#text4 {
    position: absolute;
    top: 200px;
    right: 200px;
    font-size: 48pt;
    text-shadow: 0px 1px 6px #fff,
                2px 4px 4px #0ff,
                4px 5px 6px #00f,
                0px 0px 10px #000,
                0px 0px 20px #448,
                0px 0px 30px #a4a,
                0px 0px 40px #fff;
    -webkit-transform: rotate(-90deg);
}

#text5 {
    position: absolute;
    left: 0px;
    font-size: 48pt;
    text-shadow: 2px 4px 5px #00f;
    -webkit-transform: rotate(-10deg);
}
```



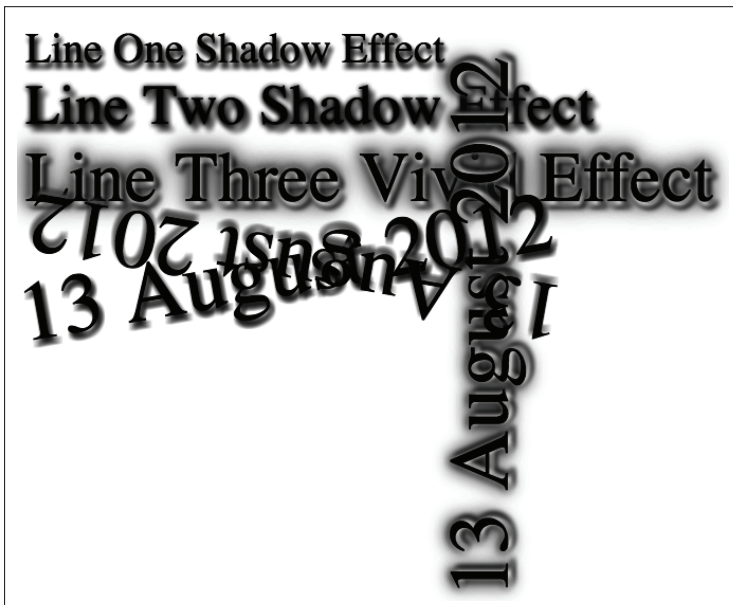
```
#text6 {
  float: left;
  font-size: 48pt;
  text-shadow: 2px 4px 5px #f00;
  -webkit-transform: rotate(-170deg);
}

/* 'transform' is explained later */
#text1:~hover, #text2:~hover, #text3:~hover,
#text4:~hover, #text5:~hover, #text6:~hover {
  -webkit-transform : scale(2) rotate(-45deg);
  transform : scale(2) rotate(-45deg);
}
```

The first selector in Listing 6.2 specifies a `font-size` of 24 and a `text-shadow` that renders text with a blue background (represented by the hexadecimal value `#00f`). The attribute `text-shadow` specifies (from left to right) the x-coordinate, the y-coordinate, the blur radius, and the color of the shadow. The second selector specifies a font size of 32 and a red shadow background (`#f00`). The third selector creates a richer visual effect by specifying multiple components in the `text-shadow` property, which were chosen by experimenting with effects that are possible with different values in the various components.

The final CSS3 selector creates an animation effect whenever users hover over any of the six text strings; the details of the animation will be deferred until later in this chapter.

Figure 6.1 displays the result of matching the selectors in the CSS stylesheet `TextShadow1.css` with the HTML `<div>` elements in the HTML page



**FIGURE 6.1** CSS3 Text Shadow Effects in a Chrome Browser on a Macbook Pro.

TextShadow1.html. The screenshot is taken in a Chrome browser on a Macbook Pro.

## CSS3 and Box Shadow Effects

You can also apply a shadow effect to a box that encloses a text string, which can be effective in terms of drawing attention to specific parts of a Web page. However, the same caveat regarding overuse applies to box shadows.



The HTML page BoxShadow1.html and BoxShadow1.css are not shown here, but they are available on the CD, and together they render a box shadow effect.

The key property is the `box-shadow` property, which has become standardized across modern browsers, so the browser-specific versions are no longer necessary:

```
#box1 {
  position: relative;
  top: 10px;
  width: 50%;
  height: 30px;
  font-size: 20px;
  box-shadow: 10px 10px 5px #800;
```

Figure 6.2 displays a screenshot of a box shadow effect in a Chrome browser on a Macbook Pro.

## CSS3 and Rounded Corners

Web developers have waited a long time for rounded corners in CSS, and CSS3 makes it very easy to render boxes with rounded corners. Listing 6.3 displays the contents of the HTML Web page RoundedCorners1.html that renders text strings in boxes with rounded corners, and Listing 6.4 displays the CSS file RoundedCorners1.css.

Line One with a Box Effect

Line Two with a Box Effect

Line Three with a Box  
Effect

FIGURE 6.2 CSS3 Box Shadow Effects in a Chrome Browser on a Macbook Pro.

**LISTING 6.3 *RoundedCorners1.html***

```

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="utf-8" />
  <title>CSS Text Shadow Example</title>
  <link href="RoundedCorners1.css" rel="stylesheet" type="text/css">
</head>

<body>
  <div id="outer">
    <a href="#" class="anchor">Text Inside a Rounded Rectangle</a>
  </div>
  <div id="text1">Line One of Text with a Shadow Effect</div>
  <div id="text2">Line Two of Text with a Shadow Effect</div>
</body>
</html>

```

Listing 6.3 contains a reference to the CSS stylesheet `RoundedCorners1.css`, which contains three CSS selectors that match the elements whose `id` attributes have the values `anchor`, `text1`, and `text2`, respectively. The CSS selectors defined in `RoundedCorners1.css` create visual effects, and as you will see, the `hover` pseudoselector enables you to create animation effects.

**LISTING 6.4 *RoundedCorners1.css***

```

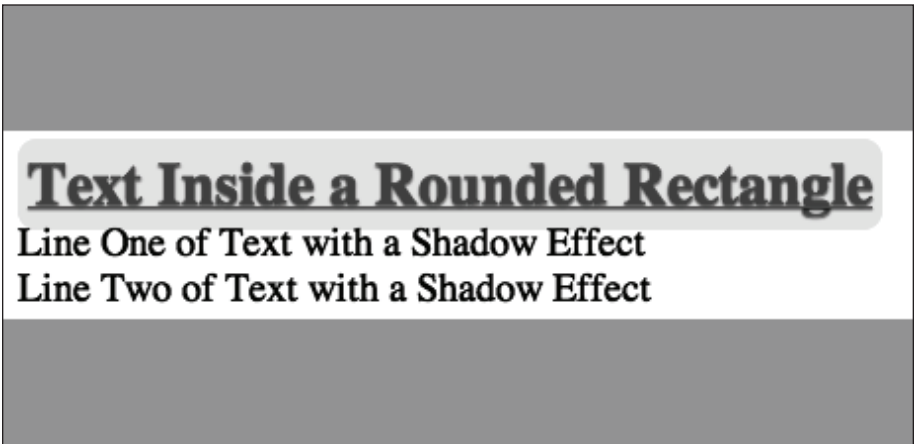
a.anchor:hover {
  background: #00F;
}

a.anchor {
  background: #FF0;
  font-size: 24px;
  font-weight: bold;
  padding: 4px 4px;
  color: rgba(255,0,0,0.8);
  text-shadow: 0 1px 1px rgba(0,0,0,0.4);
  -webkit-transition: all 2.0s ease;
  transition: all 2.0s ease;
  border-radius: 8px;
}

```

Listing 6.4 contains the selector `a.anchor:hover` that changes the text color from yellow (`#FF0`) to blue (`#00F`) during a two-second interval whenever users hover over any anchor element with their mouse.

The selector `a.anchor` contains various attributes that specify the dimensions of the box that encloses the text in the `<a>` element along with two new pairs of attributes. The first pair specifies the `transition` attribute (and a WebKit-specific prefix), which we will discuss later in this chapter. The second pair specifies the `border-radius` attribute (and the WebKit-specific



**FIGURE 6.3** CSS3 Rounded Corners Effect in a Chrome Browser on a Macbook Pro.

attribute) whose value is `8px`, which determines the radius (in pixels) of the rounded corners of the box that encloses the text in the `<a>` element. The last two selectors are identical to the selectors in Listing 6.1.

Figure 6.3 displays the result of matching the selectors that are defined in the CSS stylesheet `RoundedCorners1.css` with elements in the HTML page `RoundedCorners1.html` in a Chrome browser on a Macbook Pro.

## CSS3 GRADIENTS

CSS3 supports linear gradients and radial gradients, which enable you to create gradient effects that are as visually rich as gradients in other technologies such as SVG. The code samples in this section illustrate how to define linear gradients and radial gradients in CSS3 and then match them to HTML elements.

### Linear Gradients

CSS3 linear gradients require you to specify one or more color stops, each of which specifies a start color, an end color, and a rendering pattern. Webkit-based browsers support the following syntax to define a linear gradient:

- start point
- end point
- start color using `from()`
- zero or more stop-colors
- end color using `to()`

A start point can be specified as an  $(x, y)$  pair of numbers or percentages. For example, the pair  $(100, 25\%)$  specifies the point that is 100 pixels to the right of the origin and 25% of the way down from the top of the pattern. Recall that the origin is located in the upper left corner of the screen.

Listing 6.5 displays the contents of `LinearGradient1.html` and Listing 6.6 displays the contents of `LinearGradient1.css`, which illustrate how to use linear gradients with text strings that are enclosed in `<p>` elements and an `<h3>` element.

#### **LISTING 6.5** *LinearGradient1.html*

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8" />
  <title>CSS Linear Gradient Example</title>
  <link href="LinearGradient1.css" rel="stylesheet" type="text/css">
</head>

<body>
  <div id="outer">
    <p id="line1">line 1 with a linear gradient</p>
    <p id="line2">line 2 with a linear gradient</p>
    <p id="line3">line 3 with a linear gradient</p>
    <p id="line4">line 4 with a linear gradient</p>
    <p id="outline">line 5 with Shadow Outline</p>
    <h3><a href="#">A Line of Gradient Text</a></h3>
  </div>
</body>
</html>
```

Listing 6.5 is a simple Web page containing four `<p>` elements and one `<h3>` element. Listing 6.5 also references the CSS stylesheet `LinearGradient1.css` that contains CSS selectors that match the four `<p>` elements and the `<h3>` element in Listing 6.5.

#### **LISTING 6.6** *LinearGradient1.css*

```
#line1 {
width: 50%;
font-size: 32px;
background-image: -webkit-gradient(linear, 0% 0%, 0% 100%,
                                from(#fff), to(#f00));
background-image: -gradient(linear, 0% 0%, 0% 100%,
                            from(#fff), to(#f00));

-webkit-border-radius: 4px;
border-radius: 4px;
}

#line2 {
width: 50%;
font-size: 32px;
background-image: -webkit-gradient(linear, 100% 0%, 0% 100%,
                                from(#fff), to(#ff0));
background-image: -gradient(linear, 100% 0%, 0% 100%,
                            from(#fff), to(#ff0));

-webkit-border-radius: 4px;
border-radius: 4px;
}
```

```

#line3 {
width: 50%;
font-size: 32px;
background-image: -webkit-gradient(linear, 0% 0%, 0% 100%,
                                from(#f00), to(#00f));
background-image: -gradient(linear, 0% 0%, 0% 100%,
                             from(#f00), to(#00f));
-webkit-border-radius: 4px;
border-radius: 4px;
}

#line4 {
width: 50%;
font-size: 32px;
background-image: -webkit-gradient(linear, 100% 0%, 0% 100%,
                                from(#f00), to(#00f));
background-image: -gradient(linear, 100% 0%, 0% 100%,
                             from(#f00), to(#00f));
-webkit-border-radius: 4px;
border-radius: 4px;
}

#outline {
font-size: 2.0em;
font-weight: bold;
color: #fff;
text-shadow: 1px 1px 1px rgba(0,0,0,0.5);
}

h3 {
width: 50%;
position: relative;
margin-top: 0;
font-size: 32px;
font-family: helvetica, ariel;
}

h3 a {
position: relative;
color: red;
text-decoration: none;
-webkit-mask-image: -webkit-gradient(linear, left top, left bottom,
                                from(rgba(0,0,0,1)),
                                color-stop(50%, rgba(0,0,0,0.5)),
                                to(rgba(0,0,0,0)));
}

h3:after {
content: "This is a Line of Gradient Text";
color: blue;
}

```

The first selector in Listing 6.6 specifies a `font-size` of 32 for text, a `border-radius` of 4 (which renders rounded corners), and a linear gradient that varies from white to blue, as shown here:

```
#line1 {
width: 50%;
font-size: 32px;
background-image: -webkit-gradient(linear, 0% 0%, 0% 100%,
                                from(#fff), to(#f00));
background-image: -gradient(linear, 0% 0%, 0% 100%,
                             from(#fff), to(#f00));
-webkit-border-radius: 4px;
border-radius: 4px;
}
```

As you can see, the first selector contains two attributes with a `-webkit-` prefix and two standard attributes without this prefix. Because the next three selectors in Listing 6.6 are similar to the first selector, we will not discuss their content.

The next CSS selector creates a text outline with a nice shadow effect by rendering the text in white with a thin black shadow, as shown here:

```
color: #fff;
text-shadow: 1px 1px 1px rgba(0,0,0,0.5);
```

The final portion of Listing 6.6 contains three selectors that affect the rendering of the `<h3>` element and its embedded `<a>` element: the `h3` selector specifies the width and font size, the `h3` selector specifies a linear gradient, and the `h3:after` selector specifies the text string to display. Other attributes are specified, but these are the main attributes for these selectors.

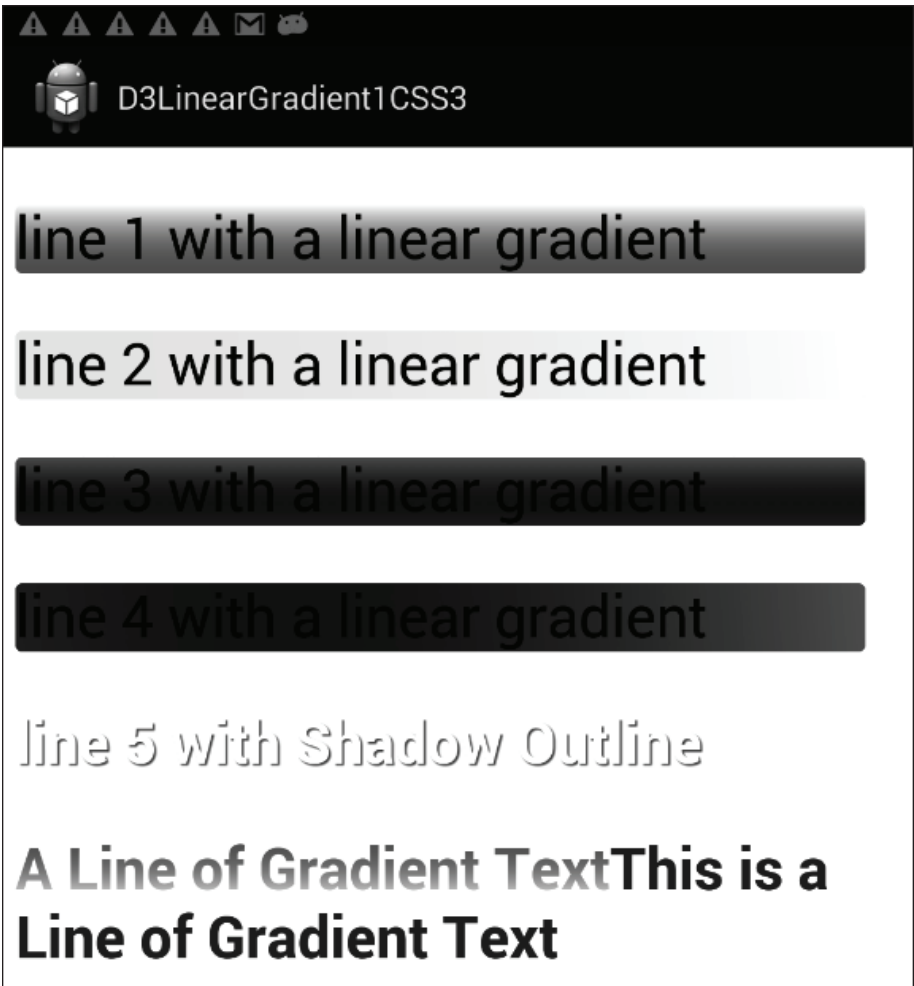
Figure 6.4 displays the result of matching the selectors in the CSS stylesheet `LinearGradient1.css` to the HTML page `LinearGradient1.html` in a landscape-mode screenshot taken from an iPad3.

## Radial Gradients

CSS3 radial gradients are more complex than CSS3 linear gradients, but you can use them to create more complex gradient effects. Webkit-based browsers support the following syntax to define a radial gradient:

- start point
- start radius
- end point
- end radius
- start color using `from()`
- zero or more color-stops
- end color using `to()`

Notice that the syntax for a radial gradient is similar to the syntax for a linear gradient, except that you also specify a start radius and an end radius.



**FIGURE 6.4** CSS3 Linear Gradient Effect on an Android Tablet.



The HTML5 Web page `RadialGradient1.html` and the CSS stylesheet `RadialGradient1.css` are not shown here, but the full listing is available on the CD. The essence of the code in the HTML5 code involves this code block:

```
<div id="outer">
  <div id="radial3">Text3</div>
  <div id="radial2">Text2</div>
  <div id="radial4">Text4</div>
  <div id="radial1">Text1</div>
</div>
```



The CSS stylesheet `RadialGradient1.css` contains five CSS selectors that match the five HTML `<div>` elements, and one of the selectors is shown here:

```
#radial1 {
background: -webkit-gradient(
    radial, 500 40%, 0, 301 25%, 360, from(red),
    color-stop(0.05, orange), color-stop(0.4, yellow),
    color-stop(0.6, green), color-stop(0.8, blue),
    to(fff)
);
}
```

The `#radial1` selector contains a `background` attribute that defines a radial gradient using the `-webkit-` prefix, and it specifies the following:

- start point of (500, 40%)
- start radius of 0
- end point of (301, 25%)
- end radius of 360
- start color of red
- end color of white (`#fff`)

The other selectors have the same syntax as the first selector, but the rendered radial gradients are significantly different. You can create these (and other) effects by specifying different start points and end points and by specifying a start radius that is larger than the end radius.

Figure 6.5 displays the result of launching `RadialGradient1.html` in a Chrome browser on a Macbook Pro.

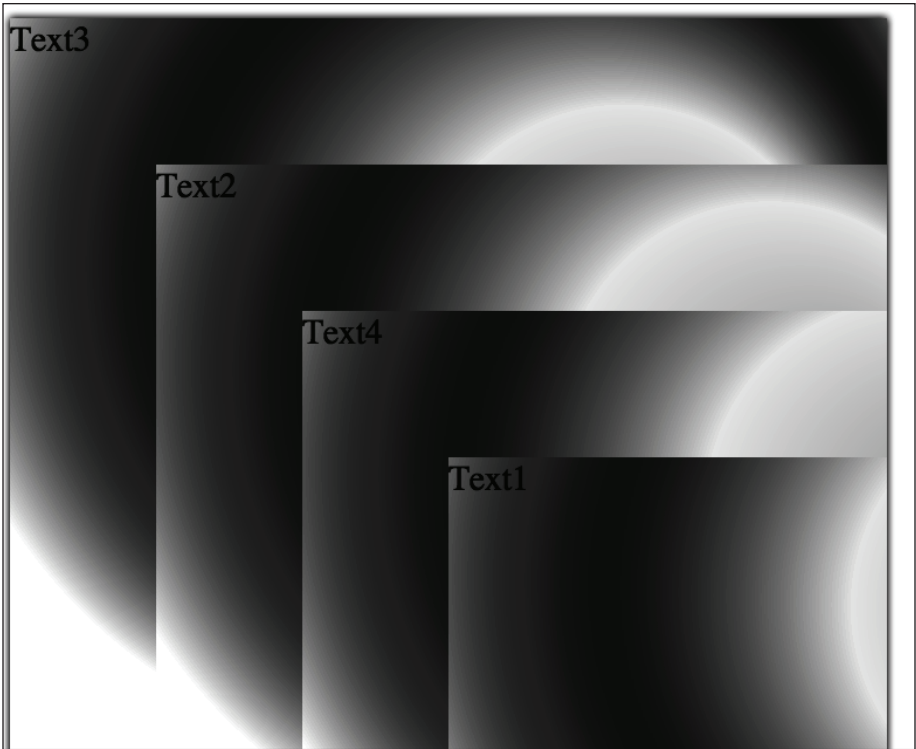
## CSS3 TWO-DIMENSIONAL TRANSFORMS

In addition to transitions, CSS3 supports four transforms that you can apply to 2D shapes and PNG files. The four CSS3 transforms are `scale`, `rotate`, `skew`, and `translate`. The following sections contain code samples that illustrate how to apply each of these CSS3 transforms to a set of PNG files. The animation effects occur when users hover over any of the PNG files; moreover, you can create partial animation effects by moving your mouse quickly between adjacent PNG files.

Listing 6.7 displays the contents of `Scale1.html` and Listing 6.8 displays the contents of `Scale1.css`, which illustrate how to scale PNG files to create a hover box image gallery.

### **LISTING 6.7** *Scale1.html*

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8" />
```



**FIGURE 6.5** CSS3 Radial Gradient Effect in a Chrome Browser on a Macbook Pro.

```

<title>CSS Scale Transform Example</title>
<link href="Scale1.css" rel="stylesheet" type="text/css">
</head>

<body>
  <header>
    <h1>Hover Over any of the Images:</h1>
  </header>

  <div id="outer">
    
    
    
    
  </div>
</body>
</html>

```

Listing 6.7 references the CSS stylesheet `Scale1.css` (which contains selectors for creating scaled effects) and four HTML `<img>` elements that reference the PNG files `sample1.png` and `sample2.png`. The remainder of Listing 6.7 is straightforward with simple boilerplate text and HTML elements.

**LISTING 6.8 *Scale1.css***

```
#outer {
float: left;
    position: relative; top: 50px; left: 50px;
}

img {
    -webkit-transition: -webkit-transform 1.0s ease;
    transition: transform 1.0s ease;
}

img.scaled {
    -webkit-box-shadow: 10px 10px 5px #800;
    box-shadow: 10px 10px 5px #800;
}

img.scaled:hover {
    -webkit-transform : scale(2);
    transform : scale(2);
}
```

The `img` selector in Listing 6.8 specifies a `transition` property that applies a transform effect that occurs during a one-second interval using the `ease` function, as shown here:

```
transition: transform 1.0s ease;
```

Next, the selector `img.scaled` specifies a `box-shadow` property that creates a reddish shadow effect (which you saw earlier in this chapter), as shown here:

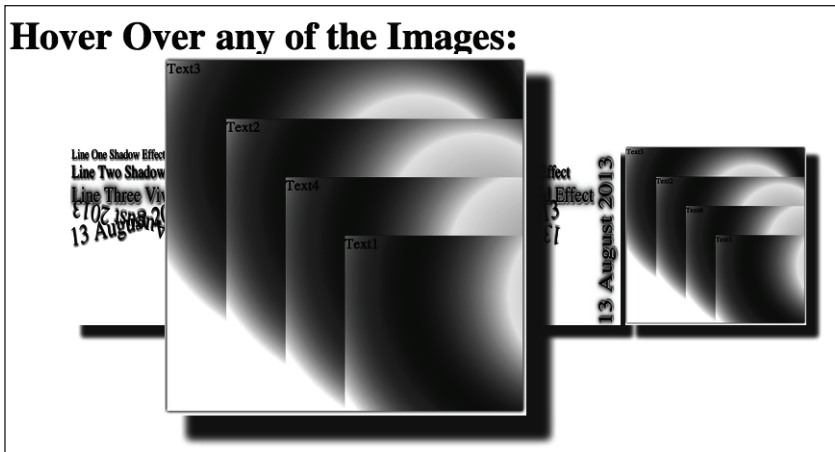
```
img.scaled {
    -webkit-box-shadow: 10px 10px 5px #800;
    box-shadow: 10px 10px 5px #800;
}
```

Finally, the selector `img.scaled:hover` specifies a `transform` attribute that uses the `scale()` function to double the size of the associated JPG file whenever users hover over any of the `<img>` elements with their mouse, as shown here:

```
transform : scale(2);
```

Because the `img` selector specifies a one-second interval using an `ease` function, the scaling effect will last for one second. Experiment with different values for the CSS3 `scale()` function and also different values for the time interval to create the animation effects that suit your needs.

Another point to remember is that you can scale both horizontally and vertically:



**FIGURE 6.6** CSS3-Based Scaling Effect on PNG Files in a Chrome Browser on a Macbook Pro.

```
img {
  -webkit-transition: -webkit-transform 1.0s ease;
  transition: transform 1.0s ease;
}

img.mystyle:hover {
  -webkit-transform : scaleX(1.5) scaleY(0.5);
  transform : scaleX(1.5) scaleY(0.5);
}
```

Figure 6.6 displays the result of matching the selectors in the CSS stylesheet `Scale1.css` to the HTML page `Scale1.html`. The landscape-mode screenshot is taken in a Chrome browser on a Macbook Pro.

## Rotate Transforms

The CSS3 transform attribute allows you to specify the `rotate()` function to create scaling effects, and its syntax looks like this:

```
rotate(someValue);
```

You can replace `someValue` with any number. When `someValue` is positive, the rotation is clockwise, when `someValue` is negative, the rotation is counter clockwise, and when `someValue` is zero, there is no rotation effect. In all cases, the initial position for the rotation effect is the positive horizontal axis.



The HTML5 Web page `Rotate1.html` and the CSS stylesheet `Rotate1.css` on the CD illustrate how to create rotation effects, a sample of which is shown here:

```
img.imageL:hover {
  -webkit-transform : scale(2) rotate(-45deg);
  transform : scale(2) rotate(-45deg);
}
```

The `img` selector that specifies a `transition` attribute that creates an animation effect during a one-second interval using the `ease` timing function, as shown here:

```
transition: transform 1.0s ease;
```

The CSS3 `transform` attribute allows you to specify the `skew()` function to create skewing effects, and its syntax looks like this:

```
skew(xAngle, yAngle);
```

You can replace `xAngle` and `yAngle` with any number. When `xAngle` and `yAngle` are positive, the skew effect is clockwise, when `xAngle` and `yAngle` are negative, the skew effect is counter clockwise, and when `xAngle` and `yAngle` are zero, there is no skew effect. In all cases, the initial position for the skew effect is the positive horizontal axis.



The HTML5 Web page `Skew1.html` and the CSS stylesheet `Skew1.css` are on the CD, and they illustrate how to create skew effects. The CSS stylesheet contains the `img` selector and specifies a `transition` attribute that creates an animation effect during a one-second interval using the `ease` timing function, as shown here:

```
transition: transform 1.0s ease;
```

There are also the four selectors `img.skewed1`, `img.skewed2`, `img.skewed3`, and `img.skewed4` that create background shadow effects with darker shades of red, yellow, green, and blue, respectively (all of which you have seen in earlier code samples).

The selector `img.skewed1:hover` specifies a `transform` attribute that performs a skew effect whenever users hover over the first `<img>` element with their mouse, as shown here:

```
transform : scale(2) skew(-10deg, -30deg);
```

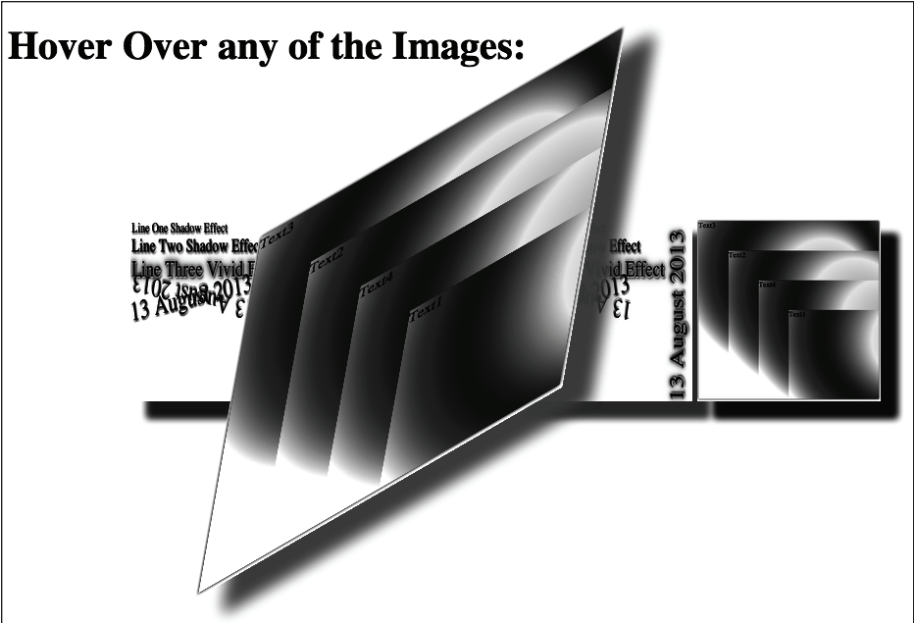
The other three CSS3 selectors also use a combination of the CSS functions `skew()` and `scale()` to create distinct visual effects. Notice that the fourth hover selector also sets the `opacity` property to `0.5`, which takes place in parallel with the other effects in this selector.

Figure 6.7 displays the result of launching the HTML page `Skew1.html` in a Chrome browser on a Macbook Pro.

The CSS3 `transform` attribute allows you to specify the `translate()` function to create an effect that involves a horizontal and/or vertical shift of an element, and its syntax looks like this:

```
translate(xDirection, yDirection);
```

## Hover Over any of the Images:



**FIGURE 6.7** CSS3-Based Skew Effects on PNG Files in a Chrome Browser on a Macbook Pro.

The translation is in relation to the origin, which is the upper left corner of the screen. Thus, positive values for `xDirection` and `yDirection` produce a shift toward the right and a shift downward, respectively, whereas negative values for `xDirection` and `yDirection` produce a shift toward the left and a shift upward; zero values for `xDirection` and `yDirection` do not cause any translation effect.



The Web page `Translate1.html` and the CSS stylesheet `Translate1.css` on the CD illustrate how to apply a translation effect (combined with other CSS3 transforms) to a JPG file.

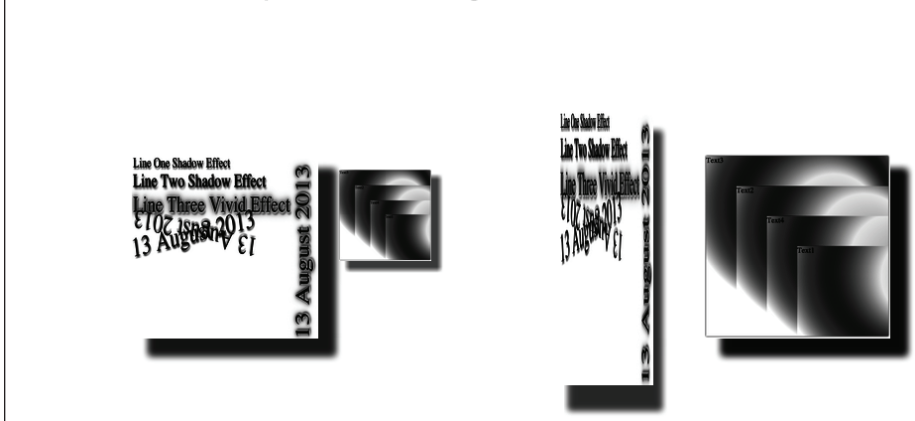
```
img.trans2:hover {
  -webkit-transform : scale(0.5) translate(-50px, -50px);
  transform : scale(0.5) translate(-50px, -50px);
}
```

The CSS stylesheet that contains the `img` selector specifies a transform effect during a one-second interval using the `ease` timing function, as shown here:

```
transition: transform 1.0s ease;
```

The four selectors `img.trans1`, `img.trans2`, `img.trans3`, and `img.trans4` create background shadow effects with darker shades of red, yellow, green, and blue, respectively, just as you saw in the previous section.

## Hover Over any of the Images:



**FIGURE 6.8** PNG Files with CSS3 Scale and Translate Effects on an iPad3.

The selector `img.trans1: hover` specifies a transform attribute that performs a scale effect and a translation effect whenever users hover over the first `<img>` element with their mouse, as shown here:

```
-webkit-transform : scale(2) translate(100px, 50px);
transform : scale(2) translate(100px, 50px);
```

Figure 6.8 displays the result of matching the selectors defined in the CSS3 stylesheet `Translate1.css` to the elements in the HTML page `Translate1.html`. The landscape-mode screenshot is taken from an iPad3.

## CSS3 THREE-DIMENSIONAL ANIMATION EFFECTS

As you know by now, CSS3 provides keyframes that enable you to create different animation effects at various points during an animation sequence. The example in this section uses CSS3 keyframes and various combinations of the CSS3 functions `scale3d()`, `rotate3d()`, and `translate3d()` to create an animation effect that lasts for four minutes.

Listing 6.9 displays the contents of `Anim240Flicker3DLGrad4.html`, which is a very simple HTML page that contains four HTML `<div>` elements.

### **LISTING 6.9** *Anim240Flicker3DLGrad4.html*

```
<!DOCTYPE html>
<html lang="en">
```

```

<head>
  <meta charset="utf-8" />
  <title>CSS3 Animation Example</title>

  <link href="Anim240Flicker3DLGrad4.css"
        rel="stylesheet" type="text/css">
</head>

<body>
  <div id="outer">
    <div id="linear1">Text1</div>
    <div id="linear2">Text2</div>
    <div id="linear3">Text3</div>
    <div id="linear4">Text4</div>
  </div>
</body>
</html>

```

Listing 6.9 is a very simple HTML5 page with CSS selectors in the corresponding CSS stylesheet that match some of the elements in Listing 6.9. As usual, the real complexity occurs in the CSS selectors that contain the code for creating the animation effects.

The CSS selectors in `Anim240Flicker3DLGrad4.css` make extensive use of the CSS3 `matrix()` function. This function requires a knowledge of matrices (which is beyond the scope of this book) to fully understand the reasons for the effects that you can create with the CSS3 `matrix()` function. If you are interested in learning about matrices, you can read the introduction to matrices (in the context of CSS3) found at <http://www.eleqtriq.com/2010/05/css-3d-matrix-transformations/>.



Because `Anim240Flicker3DLGrad4.css` is such a lengthy code sample (over six pages of selectors), only a very limited portion of the code is displayed in Listing 6.10. However, the complete code is available on the CD for this book.

#### **LISTING 6.10 A Small Section in `Anim240Flicker3DLGrad4.css`**

```

@-webkit-keyframes upperLeft {
  0% {
    -webkit-transform: matrix(1.5, 0.5, 0.0, 1.5, 0, 0)
                      matrix(1.0, 0.0, 1.0, 1.0, 0, 0);
  }
  10% {
    -webkit-transform: translate3d(50px,50px,50px)
                      rotate3d(50,50,50,-90deg)
                      skew(-15deg,0) scale3d(1.25, 1.25, 1.25);
  }
  // lots of similar code omitted
  100% {
    -webkit-transform: matrix(1.0, 0.0, 0.0, 1.0, 0, 0)
                      matrix(1.0, 0.5, 1.0, 1.5, 0, 0);
  }
}

```



```

}
// even more code omitted for brevity
#linear1 {
// minor details omitted
-webkit-animation-name: lowerLeft;
-webkit-animation-duration: 240s;
}

```

Listing 6.10 contains a Webkit-specific keyframes definition called `upperLeft` that starts with the following line:

```

@-webkit-keyframes upperLeft {
    // percentage-based definitions go here
}

```

The `#linear` selector contains properties that you have seen already, along with a property that references the keyframes identified by `lowerLeft`, and a property that specifies a duration of 240 seconds, as shown here:

```

#linear1 {
    // code omitted for brevity
    -webkit-animation-name: lowerLeft;
    -webkit-animation-duration: 240s;
}

```

Now that you know how to reference a keyframes definition in a CSS3 selector, let's look at the details of the definition of `lowerLeft`, which contains nineteen elements that specify various animation effects, but only three elements are shown in Listing 6.10.

Each element of `lowerLeft` occurs during a specific stage during the animation. For example, the eighth element in `lowerLeft` specifies the value 50%, which means that it will occur at the halfway point of the animation effect. Because the `#linear` selector contains a `-webkit-animation-duration` property whose value is 240s (shown in bold in Listing 6.10), the animation will last for four minutes, starting from the point in time when the HTML5 page is launched.

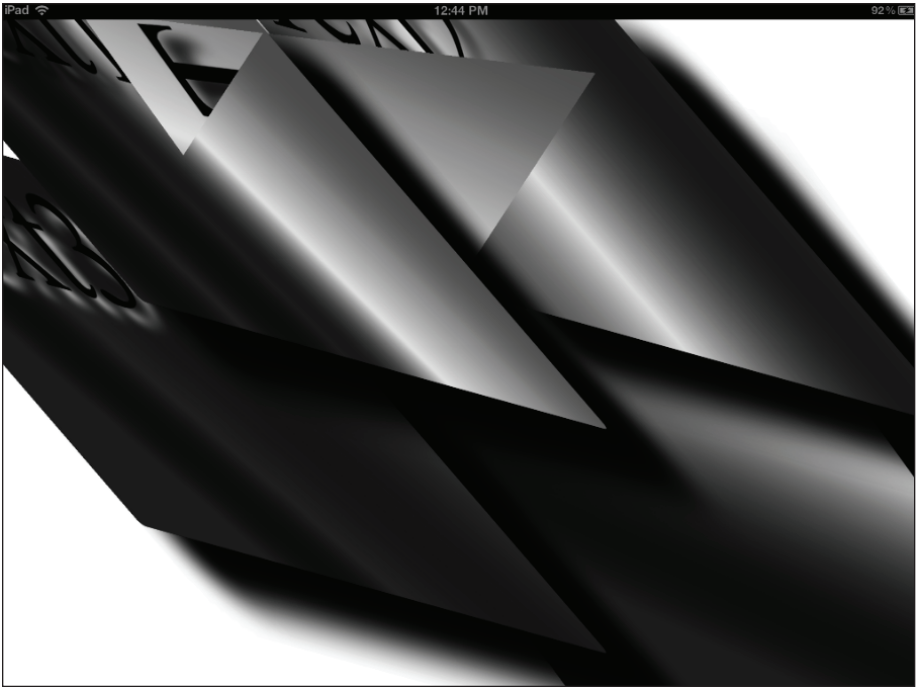
The eighth element of `lowerLeft` specifies a translation, rotation, skew, and scale effect (all of which are in three dimensions), an example of which is shown here:

```

50% {
    -webkit-transform: translate3d(250px,250px,250px)
                      rotate3d(250px,250px,250px,-120deg)
                      skew(-65deg,0) scale3d(0.5, 0.5, 0.5);
}

```

The animation effect occurs in a sequential fashion, starting with the translation, and finishing with the scale effect, which is also the case for the other elements in `lowerLeft`.



**FIGURE 6.9** CSS3 3D Animation Effects on an iPad3.

Figure 6.9 displays the initial view of matching the CSS3 selectors defined in the CSS3 stylesheet `Anim240Flicker3DLGrad4.css` with the HTML elements in the HTML page `Anim240Flicker3DLGrad4.html` in a landscape-mode screenshot taken from an iOS application running on an iPad3.

## CSS3 MEDIA QUERIES

---

CSS3 media queries are very useful logical expressions that enable you to detect mobile applications on devices with differing physical attributes and orientation. For example, with CSS3 media queries you can change the dimensions and layout of your applications so that they render appropriately on smart phones as well as tablets. Specifically, you can use CSS3 media queries in order to determine the following characteristics of a device:

- browser window width and height
- device width and height
- orientation (landscape or portrait)
- aspect ratio
- device aspect ratio
- resolution

CSS3 media queries are Boolean expressions that contain one or more simple terms (connected with `and` or `or`) that evaluate to `true` or `false`. Thus, CSS3 media queries represent conditional logic that evaluates to either `true` or `false`. As an example, the following link element loads the CSS stylesheet `mystuff.css` only if the device is a screen and the maximum width of the device is 480px:

```
<link rel="stylesheet" type="text/css"
      media="screen and (max-device-width: 480px)" href="mystuff.css"/>
```

The preceding link contains a media attribute that specifies two components: a media type of `screen` and a query that specifies a `max-device-width` whose value is 480px. The supported values for media in CSS3 media queries are `braille`, `embossed`, `handheld`, `print`, `projection`, `screen`, `speech`, `tty`, and `tv`.

The next CSS3 media query checks the media type, the maximum device width, and the resolution of a device:

```
@media screen and (max-device-width: 480px) and (resolution: 160dpi) {
  #innerDiv {
    float: none;
  }
}
```

If the CSS3 media query in the preceding code snippet evaluates to `true`, then the nested CSS selector will match the HTML element whose `id` attribute has the value `innerDiv`, and its `float` property will be set to `none` on any device whose maximum screen width is 480px. As you can see, it's possible to create compact CSS3 media queries that contain nontrivial logic, which is obviously very useful because CSS3 does not have any `if/then/else` construct that is available in other programming languages.

## CSS3 AND SVG

CSS3 selectors can reference SVG documents using the `CSS3 url()` function, which means that you can incorporate SVG-based graphic effects (including animation) in your HTML pages. As a simple example, Listing 6.10 shows you how to reference an SVG document in a CSS3 selector.

### LISTING 6.11 *Blue3DCircle1.css*

```
#circle1 {
  opacity: 0.5; color: red;
  width: 250px; height: 250px;
  position: absolute; top: 0px; left: 0px;
  font-size: 24px;
  -webkit-border-radius: 4px;
  -moz-border-radius: 4px;
```

```
border-radius: 4px;
-webkit-background: url(Blue3DCircle1.svg) top right;
-moz-background: url(Blue3DCircle1.svg) top right;
background: url(Blue3DCircle1.svg) top right;
}
```

Listing 6.11 contains various property/value pairs, and the portion containing the CSS `url()` function is shown here:

```
-webkit-background: url(Blue3DCircle1.svg) top right;
-moz-background: url(Blue3DCircle1.svg) top right;
background: url(Blue3DCircle1.svg) top right;
```

This name/value pair specifies the SVG document `Blue3DCircle1.svg` as the background for an HTML `<div>` element in an HTML5 page whose `id` attribute is `circle1`. Note that the inclusion of an attribute with a `-moz-` prefix means that this code will work in WebKit-based browsers (such as Safari and Chrome) and also in the Firefox browser.

## ADDITIONAL CODE SAMPLES ON THE CD



The CSS stylesheet `CSS3MediaQuery1.css` and the HTML5 Web page `CSS3MediaQuery1.html` illustrate how to use media queries to change the size of two images when users rotate their mobile device. The HTML Web page `TextShadow1CB1.html` and CSS stylesheet `TextShadow1CB1.css` show you how to use D3 to render a gradient-based rectangular grid that is overlaid with text strings that undergo animation effects via CSS3-based `:hover` selectors.



You can detect a change of orientation of a mobile device using simple JavaScript code, so you are not forced to use CSS3 media queries. The HTML5 Web page `CSS3OrientationJS1.html` on the CD illustrates how to use standard JavaScript to change the size of two images when users rotate their mobile device.

In essence, the code uses the value of the variable `window.orientation` to detect four different orientations of your mobile device, and in each of those four cases the dimensions of the JPG files are updated with the following type of code:

```
document.getElementById("img1").style.width = "120px";
document.getElementById("img1").style.height = "300px";
```

Although this is a very simple example, this code might give you an appreciation for the capabilities of CSS3 Media Queries.

## SUMMARY

This chapter showed you how to create graphics effects, shadow effects, and how to use CSS3 transforms in CSS3. You learned how to create animation

effects that you can apply to HTML elements, and you saw how to define CSS3 selectors to do the following:

- render rounded rectangles
- create shadow effects for text and 2D shapes
- create linear and radial gradients
- use the methods `translate()`, `rotate()`, `skew()`, and `scale()`
- create CSS3-based animation effects

The next chapter shows you how to combine D3 with CSS3, SVG, and HTML5 Canvas in HTML Web pages.

## *D3 WITH CSS3, SVG, AND HTML5 CANVAS*



This chapter presents code samples that combine D3 with HTML5-related technologies such as CSS3, SVG, and HTML5 Canvas. The intent is to provide samples that show you how to integrate these technologies with D3. You can also find code samples on the CD that show you how these technologies can coexist with D3 in the same HTML5 Web page. The code samples in this chapter assume that you have familiarized yourself with the CSS3 material in Chapter 5, the SVG material in Chapter 6, and the Appendix that covers HTML5 Canvas.

The first part of this chapter contains an HTML Web page that renders CSS3-based graphics effects with D3 followed by an example of creating CSS3-based animation effects with D3. The second part of this chapter contains an example of an HTML Web page that combines SVG with D3. The final part of this chapter shows you how to create an HTML Web page containing an HTML5 `<canvas>` element alongside D3 code.

### **D3 CODE SAMPLES WITH HTML5 CANVAS**

---

Currently, there are only a few examples available online. One nice D3 code sample that combines CSV-based data with animation effects, HTML5 Canvas, and SVG is here: <http://bl.ocks.org/syntagmatic/2411910>. An example of creating animation effects with D3 and HTML5 Canvas is here: <http://bl.ocks.org/mbostock/1276463>. A very nice example that creates animation effects with D3, HTML5 Canvas, SVG, and HTML `<div>` elements is here: <http://bl.ocks.org/sxv/4491174>. If you are interested in using HTML5 Canvas with D3, perform an Internet search for other samples that might be available.

A detailed performance comparison between SVG and HTML5 Canvas is here:

<http://trac.osgeo.org/openlayers/wiki/Future/OpenLayersAndHTML5>.

This link contains information that can help you decide when it's advantageous to use these two technologies in your custom HTML Web pages.

## UPDATED CSS3 STYLESHEETS FOR THIS CHAPTER



Many of the CSS3 stylesheets in this chapter are very similar to the CSS stylesheets that are available (as the same filename) in Chapter 6. The only changes involve replacing the occurrences of `#myCanvas` with `#svg`; the modified CSS stylesheets are available on the CD, so you do not need to make any manual modifications to these files.

## D3 AND CSS3 EFFECTS

Listing 7.1 displays the initial part of `BezierCurves1CSS3.html` that illustrates how to reference a CSS3 stylesheet to create CSS3-based graphics effects.

### LISTING 7.1 *BezierCurves1CSS3.html*

```
<!DOCTYPE html>
<head>
  <meta charset="utf-8" />
  <title>Bezier Curves and CSS3</title>
  <script src="d3.min.js"></script>
  <link href="CSS3Background6.css"
        rel="stylesheet" type="text/css">
</head>

<!-- the code below is the same as BezierCurves1.html in chapter 1 -->
<body>
  <script>
    var width = 600, height = 400, opacity=0.5;
    var cubicPath = "M20,20 C300,200 100,500 400,100";
    var quadPath = "M200,20 Q100,300 500,100";
    var fillColors = ["red", "blue"];

    // create an SVG container...
    var svgContainer = d3.select("body").append("svg")
      .attr("id", "svg")
      .attr("width", width)
      .attr("height", height);

    // create a cubic Bezier curve...
    var bezier1 = svgContainer
      .append("path")
      .attr("d", cubicPath)
      .attr("fill", fillColors[0])
      .attr("stroke", "blue")
      .attr("stroke-width", 2);

    // create a quadratic Bezier curve...
```

```

var bezier1 = svgContainer
    .append("path")
    .attr("d", quadPath)
    .attr("fill", fillColors[1])
    .attr("opacity", opacity)
    .attr("stroke", "blue")
    .attr("stroke-width", 2);

</script>
</body>
</html>

```



Listing 7.1 contains the usual boilerplate code along with an HTML `<link>` element that references the CSS stylesheet `CSS3Background6.css`, which is included on the CD for this chapter.

Because the code in the `<body>` element in Listing 7.1 is the same as the code in `BezierCurvesText1.html` in Chapter 1, it will not be discussed here. The one detail you need to notice is how Listing 7.1 sets the value of the `id` attribute to `svg`, as shown here:

```
.attr("id", "svg")
```

This detail is important because the two CSS selectors in Listing 7.2 are based on the existence of this value in Listing 7.1.

Listing 7.2 displays the initial part of `CSS3Background6.css` that contains two CSS selectors that are matched with elements in Listing 7.1.

### **LISTING 7.2** *CSS3Background6.css*

```

#svg {
position: relative; top: 0px; left: 0px;

background-color:white;
background-image:
    -webkit-radial-gradient(red 4px, transparent 48px),
    -webkit-repeating-linear-gradient(45deg, red 2px, green 4px,
                                     yellow 8px, blue 12px,
                                     transparent 16px, red 20px,
                                     blue 24px, transparent 28px,
                                     transparent 32px),
    -webkit-repeating-linear-gradient(-45deg, red 2px, green 4px,
                                     yellow 8px, blue 12px,
                                     transparent 16px, red 20px,
                                     blue 24px, transparent 28px,
                                     transparent 32px),
    -webkit-radial-gradient(blue 8px, transparent 68px);

background-size: 120px 120px, 4px 4px;
background-position: 0 0;
-webkit-box-shadow: 30px 30px 30px #000;
}

#svg:hover {
position: relative; top: 0px; left: 0px;

```



```
background-color:white;
background-image:
  -webkit-radial-gradient(red 4px, transparent 48px),
  -webkit-repeating-linear-gradient(45deg, red 5px, green 4px,
                                   yellow 8px, blue 12px,
                                   transparent 16px, red 20px,
                                   blue 24px, transparent 28px,
                                   transparent 32px),
  -webkit-repeating-linear-gradient(-45deg, red 5px, green 4px,
                                   yellow 8px, blue 12px,
                                   transparent 16px, red 20px,
                                   blue 24px, transparent 28px,
                                   transparent 32px),
  -webkit-radial-gradient(blue 8px, transparent 68px);

background-size: 120px 120px, 4px 4px;
background-position: 0 0;
}
```

Listing 7.2 contains two CSS selectors, and although they might look complicated, their purpose is actually simple. The first selector matches the dynamically created HTML `<div>` element in Listing 7.1. The second CSS selector matches whenever users hover over this same element with their mouse. As for the details of the contents of the two selectors (which are very similar), please read the relevant section in Chapter 6 (the CSS chapter).

## D3 AND CSS3 ANIMATION EFFECTS

Listing 7.3 displays `BezierCurvesAndText1CSS3Animation1.html` that illustrates how to reference a CSS3 stylesheet that creates animation effects.

### **LISTING 7.3** *BezierCurvesAndText1CSS3Animation1.html*

```
<!DOCTYPE html>
<head>
  <meta charset="utf-8" />
  <title>Bezier Curves, Text, and CSS3 Animation</title>
  <script src="d3.min.js"></script>
  <link href="HoverAnimation2.css"
        rel="stylesheet" type="text/css">
</head>

<!-- same as BezierCurves1AndText1.html in chapter 1 -->
<body>
  <script>
    var width = 600, height = 400, opacity=0.5;
    var cubicPath = "M20,20 C300,200 100,500 400,100";
    var quadPath = "M200,20 Q100,300 500,100";

    var cValues = "M20,20 C300,200 100,500 400,100";
    var qValues = "M200,20 Q100,300 500,100";
```

```

var fontSizeC = "24";
var fontSizeQ = "18";

var textC =
  "Sample Text that follows a path of a cubic Bezier curve";

var textQ =
  "Sample Text that follows a path of a quadratic Bezier curve";

var fillColors = ["red", "blue", "green", "yellow"];

// create an SVG container...
var svgContainer = d3.select("body")
  .append("svg")
  .attr("id", "svg")
  .attr("width", width)
  .attr("height", height);

var defs =
  = svgContainer
    .append("svg:defs");

var patternC = defs.append("svg:path")
  .attr("id", "pathInfoC")
  .attr("d", cValues);

var patternQ = defs.append("svg:path")
  .attr("id", "pathInfoQ")
  .attr("d", qValues);

// now add the 'g' element...
var g1 = svgContainer.append("svg:g");

// create a cubic Bezier curve...
var bezierC = g1.append("path")
  .attr("d", cubicPath)
  .attr("fill", fillColors[0])
  .attr("stroke", "blue")
  .attr("stroke-width", 2);

// text following a cubic Bezier curve...
var textC = g1.append("text")
  .attr("id", "textStyleC")
  .attr("stroke", "blue")
  .attr("fill", fillColors[1])
  .attr("stroke-width", 2)
  .append("textPath")
  .attr("font-size", fontSizeC)
  .attr("xlink:href", "#pathInfoC")
  .text(textC);

// create a quadratic Bezier curve...
var bezierQ = g1.append("path")
  .attr("d", quadPath)
  .attr("fill", fillColors[1])
  .attr("opacity", opacity)

```

```

        .attr("stroke", "blue")
        .attr("stroke-width", 2);

    // text following a cubic Bezier curve...
    var textQ = gl.append("text")
        .attr("id", "textStyleQ")
        .attr("stroke", fillColors[3])
        .attr("stroke-width", 2)
        .append("textPath")
        .attr("font-size", fontSizeQ)
        .attr("xlink:href", "#pathInfoQ")
        .text(textQ);

    </script>
</body>
</html>

```

Listing 7.3 starts with the usual boilerplate code along with an HTML `<link>` element to reference the CSS3 stylesheet `HoverAnimation2.css`. Because this CSS3 stylesheet contains CSS3 transforms that you have already seen in Chapter 6, its contents are not discussed here, but the entire code listing is available on the CD. Similarly, the code in the `<body>` element is essentially the same as the code contained in the HTML Web page `Bezier-Curves1AndText1.html` in Chapter 1, so it will not be discussed here.



## D3 AND HTML5 CANVAS

The code sample in this chapter shows you how to combine D3 code with HTML5 Canvas code in the same HTML Web page. Although HTML5 Canvas is not covered in any of the chapters, you can read material in the appropriate appendix on the CD to acquaint yourself with some of the HTML5 APIs that are available.



Listing 7.4 displays the contents of `D3AndHTML5Canvas1.html` that contain graphics created with D3 code as well as graphics created with HTML5 Canvas code.

### **LISTING 7.4** *D3AndHTMLCanvas1.html*

```

<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8" />
    <title>D3 and HTML5 Canvas</title>
    <script src="d3.min.js"></script>

    <style>
      input {
        width:350px;
        font-size:16px;
        background-color:#f00;
      }
    </style>
  </head>

```

```

<body>
<script>
  var width = 700, height = 200, opacity=0.5;
  var cubicPath  = "M20,0 C300,200 100,260 400,100";
  var quadPath   = "M200,0 Q100,260 500,100";
  var fillColors = ["red", "blue"];

  // create an SVG container...
  var svgContainer = d3.select("body").append("svg")
    .attr("id", "svg")
    .attr("width", width)
    .attr("height", height);

  // create a cubic Bezier curve...
  var bezier1 = svgContainer
    .append("path")
    .attr("d", cubicPath)
    .attr("fill", fillColors[0])
    .attr("stroke", "blue")
    .attr("stroke-width", 2);

  // create a quadratic Bezier curve...
  var bezier1 = svgContainer
    .append("path")
    .attr("d", quadPath)
    .attr("fill", fillColors[1])
    .attr("opacity", opacity)
    .attr("stroke", "blue")
    .attr("stroke-width", 2);
</script>

<script><!--
window.addEventListener('load', function () {
  // Get the canvas element
  var elem = document.getElementById('myCanvas');
  if (!elem || !elem.getContext) {
    return;
  }

  // Get the canvas 2d context.
  var context = elem.getContext('2d');
  if (!context) {
    return;
  }

  var basePointX = 10, basePointY = 10,
      clickCount = 0, lineCount = 128,
      lineLength = 200, lineWidth = 4,
      lineColor = "";

  var hexArray = new Array('0','1','2','3','4','5','6','7',
    '8','9','a','b','c','d','e','f');

  redrawCanvas = function() {

```

```

        // clear the canvas before drawing new set of rectangles
        //context.clearRect(0, 0, elem.width, elem.height);

        // first set of line segments...
        for(var y=0; y<lineCount; y++) {
            context.beginPath();

            lineColor = '#' + hexArray[y%16] + '00';
            context.strokeStyle = lineColor;
            context.lineCap = 'square'; // 'butt' 'round' 'square'
            context.lineJoin = 'round'; // 'bevel' 'miter' 'round'
            context.lineWidth = lineWidth;

            context.moveTo(basePointX, basePointY+y);
            context.lineTo(basePointX+lineLength, basePointY+y);
            context.stroke();
        }

        // second set of line segments...
        for(var y=0; y<lineCount; y++) {
            context.beginPath();

            lineColor = '#' + hexArray[y%16] + hexArray[y%16] + '0';
            context.strokeStyle = lineColor;

            context.moveTo(basePointX+lineLength, basePointY+y);
            context.lineTo(basePointX+2*lineLength, basePointY+y);
            context.stroke();
        }

        // third set of line segments...
        for(var y=0; y<lineCount; y++) {
            context.beginPath();

            lineColor = '#00' + hexArray[y%16];
            context.strokeStyle = lineColor;

            context.moveTo(basePointX+2*lineLength, basePointY+y);
            context.lineTo(basePointX+3*lineLength, basePointY+y);
            context.stroke();
        }

        ++clickCount;
        basePointX += 1;
        basePointY += 1;
    }

    // render the line segments
    redrawCanvas();
}, false);
// --></script>

<div>
<canvas id="myCanvas" width="700" height="200">No support for Canvas
    alt="Example Rendering Line Segments.">
</canvas>
</div>

```

```

<div>
  <input type="button" onclick="redrawCanvas();return false"
        value="Redraw the Line Segments" />
</div>
</body>
</html>

```

Listing 7.4 starts with boilerplate code and a simple `<style>` element with some styling definitions. The HTML `<body>` element contains two `<script>` elements. The first `<script>` element contains D3 code that renders a quadratic Bezier curve and a cubic Bezier curve, the details of which you have seen in previous code samples in Chapter 1.

The first key point to notice about the second `<script>` element is that it contains HTML5 Canvas code for rendering three sets of line segments. The first portion of code verifies that there is an HTML5 `<canvas>` element in the `<body>` element of Listing 7.4 (which does exist in this code sample). After initializing some JavaScript variables, Listing 7.4 contains three loops for rendering three different sets of line segments with gradient shading. The first loop is reproduced here:

```

// first set of line segments...
for(var y=0; y<lineCount; y++) {
  context.beginPath();

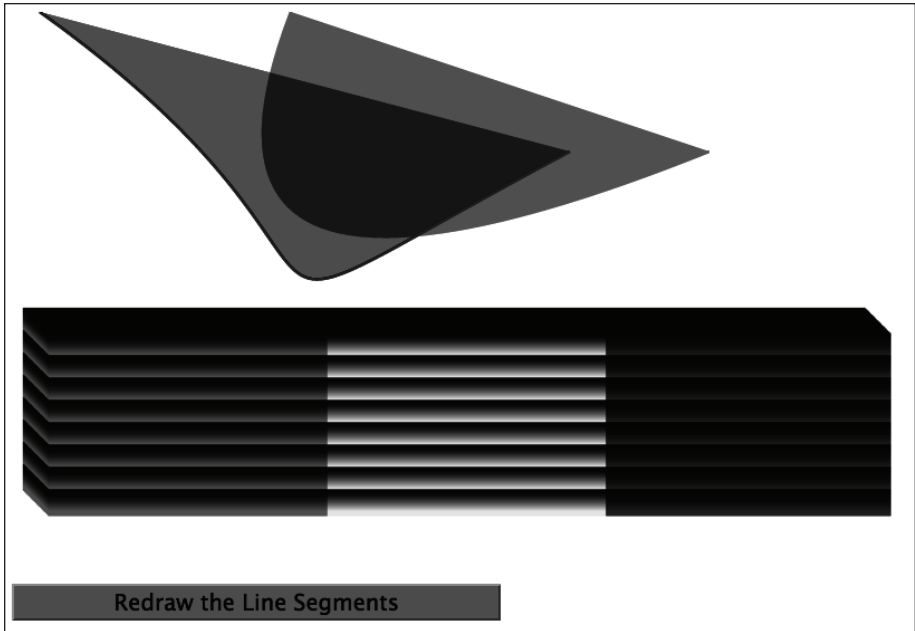
  lineColor = '#' + hexArray[y%16] + '00';
  context.strokeStyle = lineColor;
  context.lineCap = 'square'; // 'butt' 'round' 'square'
  context.lineJoin = 'round'; // 'bevel' 'miter' 'round'
  context.lineWidth = lineWidth;

  context.moveTo(basePointX, basePointY+y);
  context.lineTo(basePointX+lineLength, basePointY+y);
  context.stroke();
}

```

The preceding loop uses the JavaScript array `hexArray` to compute the color of each line segment then sets some attributes for the line segment, and then it uses the HTML5 Canvas `moveTo()` and `lineTo()` methods to set the position of the line segment. Notice that this differs from the manner in which D3 (and SVG) renders a line segment. The second and third loops perform similar effects but with different gradient shading.

The second key point to notice is that the three loops are part of a JavaScript function called `redrawCanvas`, which is conveniently defined inside an event handler that is executed during a load event (i.e., when the HTML Web page is loaded into a browser). The third key point to notice that the final code snippet is `redrawCanvas()`, which in turn executes the code in the JavaScript function, thereby rendering the HTML5 Canvas graphics.



**FIGURE 7.1** D3 Graphics and HTML5 Canvas Graphics in the same HTML Web Page.

Figure 7.1 displays a pair of Bezier curves in D3 and a set of line segments rendered in HTML5 Canvas.

## D3 AND SVG

Listing 7.5 displays the contents of `D3AndSVG1.html` that illustrates how to generate D3-based graphics and also reference an SVG document inside this HTML Web page. Listing 7.5 displays the contents of `ArchEllipses1.svg`, which is referenced at the bottom of Listing 7.5.

### **LISTING 7.5** *D3AndSVG1.html*

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8" />
    <title>D3 and Embedded SVG</title>
    <script src="d3.min.js"></script>
  </head>

  <body>
    <script>
      var width = 600, height = 200, opacity=0.5;
      var cubicPath = "M20,0 C300,200 100,260 400,100";
```

```

var quadPath    = "M200,0 Q100,260 500,100";
var fillColors = ["red", "blue"];

// create an SVG container...
var svgContainer = d3.select("body").append("svg")
    .attr("id", "svg")
    .attr("width", width)
    .attr("height", height);

// create a cubic Bezier curve...
var bezier1 = svgContainer
    .append("path")
    .attr("d", cubicPath)
    .attr("fill", fillColors[0])
    .attr("stroke", "blue")
    .attr("stroke-width", 2);

// create a quadratic Bezier curve...
var bezier1 = svgContainer
    .append("path")
    .attr("d", quadPath)
    .attr("fill", fillColors[1])
    .attr("opacity", opacity)
    .attr("stroke", "blue")
    .attr("stroke-width", 2);

</script>

<div id="ArchEllipses1">
  <embed id="embed1" src="ArchEllipses1.svg"
    width="600" height="250" type="image/svg+xml">
  </embed>
</div>
</body>
</html>

```

Listing 7.5 starts with boilerplate code followed by a `<script>` element that defines two Bezier curves (which is very familiar to you by now). The new concept that is introduced in this code sample is the final `<div>` element (shown in bold in Listing 7.5) that is located at the bottom of the `<body>` element, as shown here:

```

<div id="ArchEllipses1">
  <embed id="embed1" src="ArchEllipses1.svg"
    width="600" height="250" type="image/svg+xml">
  </embed>
</div>

```

The preceding `<div>` element contains an HTML `<embed>` element, which in turn references the SVG document `ArchEllipses1.svg`. As you can probably guess, the effect of including this HTML `<embed>` element is to render the SVG graphics that are defined in `ArchEllipses1.svg` whose contents are shown in Listing 7.6.



**LISTING 7.6 ArchEllipses1.svg**

```

<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
    "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">

<svg xmlns="http://www.w3.org/2000/svg"
    xmlns:xlink="http://www.w3.org/1999/xlink"
    onload="init(evt)"
    width="100%" height="100%">

<script>
<![CDATA[
    var basePointX      = 250,   basePointY      = 200,
        currentX        = 0,     currentY        = 0,
        offsetX         = 0,     offsetY         = 0,
        radius          = 0,     minorAxis       = 60,
        majorAxis       = 30,    spiralCount     = 4,
        Constant        = 0.25, angle           = 0,
        maxAngle        = 720,   angleDelta      = 2,
        strokeWidth    = 1;

    var redColor = "rgb(255,0,0)";
    var ellipseNode= null, svgDocument=null;
    var target= null, gcNode=null;
    var svgNS = "http://www.w3.org/2000/svg";

    function init(evt) {
        svgDocument = event.target.ownerDocument;
        gcNode = svgDocument.getElementById("gc");
        drawSpiral(event);
    }

    function drawSpiral(evt) {
        for(angle=0; angle<maxAngle; angle+=angleDelta) {
            radius = Constant*angle;
            offsetX = radius*Math.cos(angle*Math.PI/180);
            offsetY = radius*Math.sin(angle*Math.PI/180);
            currentX = basePointX+offsetX;
            currentY = basePointY-offsetY;

            ellipseNode =
                svgDocument.createElementNS(svgNS, "ellipse");

            ellipseNode.setAttribute("fill", redColor);
            ellipseNode.setAttribute("stroke-width", strokeWidth);

            if( angle % 3 == 0 ) {
                ellipseNode.setAttribute("stroke", "yellow");
            } else {
                ellipseNode.setAttribute("stroke", "green");
            }

            ellipseNode.setAttribute("cx", currentX);
            ellipseNode.setAttribute("cy", currentY);
            ellipseNode.setAttribute("rx", majorAxis);
            ellipseNode.setAttribute("ry", minorAxis);

```

```

        gcNode.appendChild(ellipseNode);
    }
} // drawSpiral
]]> </script>

<!-- ===== -->
<g id="gc" transform="scale(1.0, 0.5)">
    <rect x="0" y="0"
        width="800" height="500"
        fill="none" stroke="none"/>
</g>
</svg>

```

There are several key points to notice about the code in Listing 7.6, which starts with some boilerplate code and the definition of the SVG `<svg>` element. Notice the code snippet (shown in bold) in Listing 7.6, which is reproduced here:

```
onload="init(evt)"
```

The purpose of the preceding code snippet is to execute the JavaScript `init()` function that is defined in Listing 7.6 when the SVG document is loaded into a browser.

The second key point to notice is that JavaScript code in an SVG document must be defined in a so-called Character DATA (CDATA) section, which in turn is placed inside a `<script>` element. In Listing 7.6, the CDATA section starts by defining some JavaScript variables followed by the JavaScript `init()` function, as shown here:

```

function init(evt) {
    svgDocument = event.target.ownerDocument;
    gcNode = svgDocument.getElementById("gc");
    drawSpiral(event);
}

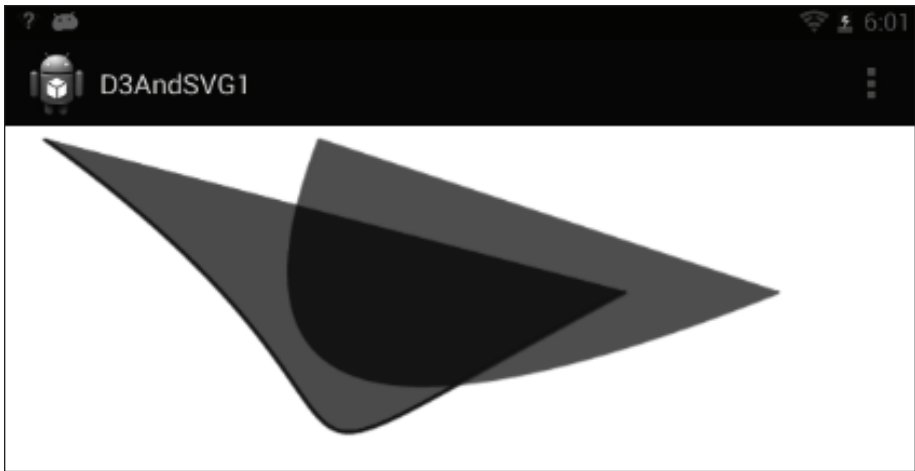
```

The preceding function contains three important lines of code. First, there is a reference to the current SVG document. Second, there is a reference to the SVG `<gc>` element (defined at the bottom of Listing 7.6). Third, it invokes the JavaScript `drawSpiral()` function that creates the graphics effect in SVG.

The JavaScript function `drawSpiral()` contains a loop that creates a set of SVG `<ellipse>` elements that follow the points on an Archimedean spiral (which is a famous polar equation). In SVG, you create an SVG `<ellipse>` element by invoking a namespace-qualified method (which also requires a namespace as a parameter), as shown in this code snippet:

```
ellipseNode = svgDocument.createElementNS(svgNS, "ellipse");
```

As you already know from Chapter 1, you need to specify values for the `cx`, `cy`, `rx`, and `ry` attributes of an SVG ellipse, which is also performed in the loop in Listing 7.6.



**FIGURE 7.2** D3 Graphics and SVG Graphics in the Same HTML Web Page.

The final code snippet in the loop is the code that appends each new SVG `<ellipse>` element to the current DOM:

```
gcNode.appendChild(ellipseNode);
```

Figure 7.2 displays a pair of Bezier curves in D3 and an Archimedean spiral from an SVG document.

## BUBBLE CHARTS WITH JSON DATA

The code sample in this section shows you how to render a bubble chart using JSON-based data that contains a hierarchical structure (representing a tree) for the data.

A basic bubble chart with JSON-based data is available online here:

<http://jsfiddle.net/jdias/LCJe6/>

The code sample in Listing 7.7 makes some modifications and enhancements to the online code sample.

### **LISTING 7.7** *BubbleChart2.html*

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8" />
    <title>Bubble Chart with CSS3</title>

    <link href="CSS3BubbleChart2.css"
          rel="stylesheet" type="text/css">

    <script src="d3.min.js"></script>
  </head>
```

```

<body>
<script>
  var items = {className:"", children:[
    {className:"TVI",  packageName:'aa', value:2000},
    {className:"SIC",  packageName:'aa', value:1520},
    {className:"RTP1", packageName:'bb', value: 800},
    {className:"RTP2", packageName:'bb', value: 150}
  ]};

  var r = 400,
      format = d3.format(",d"),
      fill = d3.scale.category10();

  var bubble = d3.layout.pack()
    .sort(null)
    .size([r, r]);

  var chart = d3.select("body")
    .append("svg:svg")
    .attr("id", "svg")
    .attr("width", r)
    .attr("height", r)
    .attr("class", "bubble");

  var node = chart.selectAll("g.node")
    .data(bubble.nodes(items))
    .enter()
    .append("svg:g")
    .attr("class", "node")
    .attr("transform", function(d) {
      return "translate(" + d.x + "," + d.y + ")";
    });

  node.append("svg:title")
    .text(function(d) {
      return d.className + ": " + format(d.value);
    });

  node.append("svg:circle")
    .attr("r", function(d) { return d.r; })
    .style("fill", function(d) {
      return fill(d.packageName);
    });

  node.append("svg:text")
    .attr("text-anchor", "middle")
    .attr("dy", ".3em")
    .text(function(d) {
      return d.className.substring(0, d.r/3);
    });
</script>
</body>
</html>

```

The code in Listing 7.7 makes the following enhancements:

- adding a CSS3 stylesheet with gradient effects
- adding an `id` attribute to the `<svg>` element
- adding an `svg:hover` selector to the `<svg>` element

Listing 7.8 displays the first selector (the second one is very similar) in the CSS stylesheet `CSS3BubbleChart2.css` that matches elements in Listing 7.7.

**LISTING 7.8 CSS3BubbleChart2.css**

```
#svg {
position: relative; top: 10px; left: 10px;

background-color:white;
background-image:
  -webkit-radial-gradient(red 4px, transparent 48px),
  -webkit-repeating-linear-gradient(45deg, red 2px, green 4px,
                                   yellow 8px, blue 12px,
                                   transparent 16px, red 20px,
                                   blue 24px, transparent 28px,
                                   transparent 32px),
  -webkit-repeating-linear-gradient(-45deg, red 2px, green 4px,
                                   yellow 8px, blue 12px,
                                   transparent 16px, red 20px,
                                   blue 24px, transparent 28px,
                                   transparent 32px),
  -webkit-radial-gradient(blue 8px, transparent 68px);

background-size: 120px 120px, 4px 4px;
background-position: 0 0;
-webkit-box-shadow: 30px 30px 30px #000;
}
```

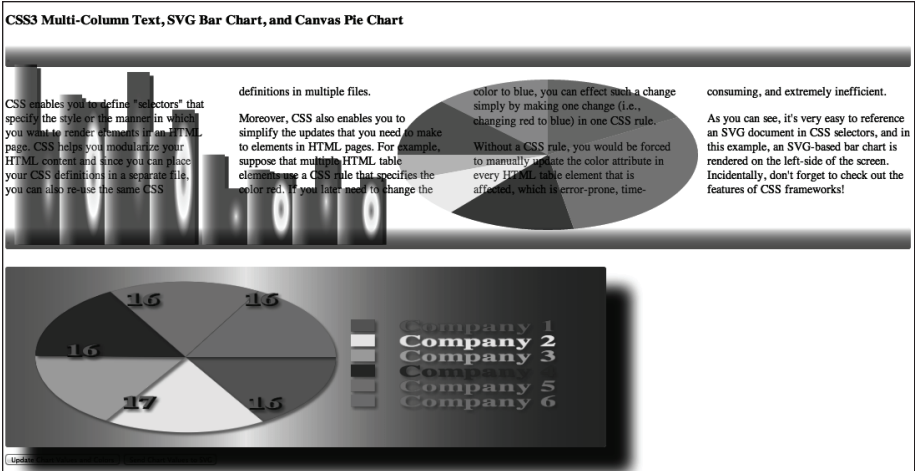
Figure 7.3 displays a bubble chart based on hierarchical JSON-formatted data in a Chrome browser on a Macbook Pro.

## ADDITIONAL CODE SAMPLES ON THE CD



The CD contains the HTML Web pages `D3AndSVG1BlurFilter1.html` and `D3AndSVG1Pattern1.html` that are based on `D3AndSVG1.html` discussed in this chapter. The first code sample adds an SVG `<filter>` element (that specifies a Gaussian blur filter), and the second code sample adds an SVG `<pattern>` element. The CD also contains an integrated example that combines HTML5 Canvas, CSS3, and SVG in the same HTML5 Web page. There are five files for this coexistence code sample:

```
MouseCSS3SVGCanvas1.html
MouseCSS3SVGCanvas1.css
MouseCSS3SVGCanvas1.svg
MouseCSS3EllipticPieChart1.svg
CSS3Background1.css
```



**FIGURE 7.3** A Bubble Chart with JSON Data in a Chrome Browser on a Macbook Pro.

## SUMMARY

This chapter started with an example of combining D3 with HTML5 Canvas followed by examples of combining D3 with CSS3 selectors. Next, you learned how to combine D3 with CSS3-based animation effects, and you learned how to combine D3 with SVG the same HTML Web page. Finally, you saw an example of creating bubble charts with JSON data. The next chapter shows you how to use D3 with Ajax and HTML5 WebSockets.



## *D3 WITH AJAX, HTML5 WEBSOCKETS, AND NODEJS*

This chapter contains an overview of several technologies that you can use in conjunction with D3, such as Ajax, HTML5 WebSockets, and NodeJS. The first part of this chapter shows you how to create Web pages with D3 and Ajax and how to get dynamically generated data from a PHP program and from a MySQL database. The second part of this chapter contains some examples of using D3 with HTML5 WebSockets. If you are unfamiliar with WebSockets, you can read the appendix, which will help you understand the code in this section. You can also visit this Website: <http://www.websocket.org>. The third part of this chapter contains some examples of using D3 with NodeJS. If you are unfamiliar with NodeJS, you can read the appendix, which will help you understand the code in this section.

### **D3 AND AJAX REQUESTS**

---

Many toolkits are available that provide Ajax support, and if you need this functionality in D3, there are at least two methods for making Ajax-based requests. One method provides generic Ajax support, and the second method provides support for JSON-based data, as shown here:

```
d3.xhr( requestURL, callback);  
d3.json( requestURL, callback);
```

Listing 8.1 displays the contents of `BarChartWeatherData.html` that illustrate how to use `d3.xhr()` to retrieve a data set for populating a bar chart. The data values are just a set of random numbers, but let's make the assumption that the data represents a set of temperature values. This code sample is longer than most of the other code samples in this book, and we will focus on the portions that contain new D3 concepts.



**LISTING 8.1** *BarChartWeatherData.html*

```

<!DOCTYPE
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>Ajax for a Bar Chart </title>
  <script src="d3.min.js"></script>
</head>

<body>
  <p>Click anywhere to update the bar chart:</p>

  <script>
    var width=600, height=300;
    var offsetY=15, roundBand=0.05, theColor="";
    var clickCount=0, weight=0, multiplier=100;
    var textColor="white", jsonData="";

    var randomNumbersURL =
"http://www.random.org/integers/?num=10&min=1&max=6&col=1&base=10&
format=plain&rnd=new";

    // initial values for the bar chart...
    var dataValues = [20, 15, 30, 25, 12, 18, 28, 38,
                      21, 32, 26, 20, 18, 8, 16, 18];

    var dataCount = dataValues.length;

    var xScale = d3.scale.ordinal()
      .domain(d3.range(dataValues.length))
      .rangeRoundBands([0, width], roundBand);

    var yScale = d3.scale.linear()
      .domain([0, d3.max(dataValues)])
      .range([0, height]);

    //Create SVG element
    var svg = d3.select("body")
      .append("svg")
      .attr("width", width)
      .attr("height", height);

    // Create bar elements...
    svg.selectAll("rect")
      .data(dataValues)
      .enter()
      .append("rect")
      .attr("x", function(d, i) {
        return xScale(i);
      })
      .attr("y", function(d) {
        return height - yScale(d);
      })
      .attr("width", xScale.rangeBand())
      .attr("height", function(d) {
        return yScale(d);

```

```

    })
    .attr("fill", function(d) {
        return "rgb(0, 0, " + (d * 10) + ")";
    });

// Create bar labels...
svg.selectAll("text")
    .data(dataValues)
    .enter()
    .append("text")
    .text(function(d) {
        return d;
    })
    .attr("text-anchor", "middle")
    .attr("x", function(d, i) {
        return xScale(i) + xScale.rangeBand()/2;
    })
    .attr("y", function(d) {
        return height - yScale(d) + offsetY;
    })
    .attr("font-family", "sans-serif")
    .attr("font-size", "12px")
    .attr("fill", textColor);

// update bar chart with web service values...
svg.on("click", function() {
    ++clickCount;

    // OBTAIN NEW DATA VALUES VIA d3.xhr() HERE:
    d3.xhr(randomNumbersURL, processWeatherData);

    // put the data values in an array
    function processWeatherData(data) {
        var dataValues = [];

        var intRegex = /^\d+$/;
        for (var key in data) {
            var obj = data[key];
            if (intRegex.test(obj)) {
                dataValues.push(obj);
            }
        }

        for (var x=0; x<dataValues.length; x++) {
            console.log("integer: "+dataValues[x]);
        }
    }

    // reset the scaled x and y values...
    xScale = d3.scale.ordinal()
        .domain(d3.range(dataValues.length))
        .rangeRoundBands([0, width], roundBand);

    yScale = d3.scale.linear()
        .domain([0, d3.max(dataValues)])
        .range([0, height]);

```

```

// Update the rectangles...
svg.selectAll("rect")
  .data(dataValues)
  .attr("y", function(d) {
    return height - yScale(d);
  })
  .attr("height", function(d) {
    return yScale(d);
  })
  .attr("fill", function(d) {
    weight = Math.floor((d*10) % 255);
    colorR = "rgb("+weight+", 0, 0)";
    colorG = "rgb(0,"+weight+", 0)";
    colorB = "rgb(0, 0,"+weight+")";

    if(clickCount % 3 == 0) {
      theColor = colorR;
    } else if(clickCount % 3 == 1) {
      theColor = colorG;
    } else if(clickCount % 3 == 2) {
      theColor = colorB;
    }

    return theColor;
  });

// Update the labels...
svg.selectAll("text")
  .data(dataValues)
  .text(function(d) {
    //return d;
    return Math.floor(d);
  })
  .attr("x", function(d, i) {
    return xScale(i) + xScale.rangeBand()/2;
  })
  .attr("y", function(d) {
    return height - yScale(d) + offsetY;
  });
});
</script>
</body>
</html>

```

Listing 8.1 consists of the following sections of code:

```

// initialize JavaScript variables
// Create SVG element
// Create bar elements
// Create bar labels
// update bar chart after a click event

```

If you review the code in Listing 8.1, every section of code has been covered in code samples in earlier chapters. The only exception is the following

JavaScript code block that parses the returned values from the Web service to extract only data that represents a number:

```
// put the data values in an array
function processWeatherData(data) {
    var dataValues = [];

    // a regular expression for positive numbers
    var intRegex = /^d+$/;
    for (var key in data) {
        var obj = data[key];
        if (intRegex.test(obj)) {
            dataValues.push(obj);
        }
    }

    for (var x=0; x<dataValues.length; x++) {
        console.log("integer: "+dataValues[x]);
    }
}
```

The preceding code block specifies the following regular expression that matches a string consisting only of the digits 0 through 9:

```
var intRegex = /^d+$/;
```

The subsequent loop tests each line of data against the preceding regular expression and appends the line of data to an array only if the data is a bona fide positive number.

## D3 WITH PHP DATA

This section provides a PHP script that generates a set of random numbers (in much the same way as the previous example) and an HTML Web page with D3 code that invokes the PHP script to retrieve the list of random numbers.

Listing 8.2 only shows you the D3 code to invoke the PHP script without rendering a specific chart or graph. However, you can use the code in Listing 8.1 (or some other HTML Web page of your choice) and then replace the `d3.xhr()` code block with the code in Listing 8.2. This section uses a XAMPP (“X (as in “cross-platform”), Apache, MySQL, PHP, Perl”) installation on a Macbook Pro, but you can use a separate installation of Apache if you prefer.

Listing 8.2 displays the contents of the HTML Web page `PHPRandomData1.html` that invokes the PHP script in Listing 8.3 to retrieve data that you can use to render a chart or graph.

### **LISTING 8.2** *PHPRandomData1.html*

```
<!DOCTYPE HTML>
<html>
<head>
    <meta charset="utf-8">
```

```

<title>Data via PHP</title>

<script src="d3.min.js"></script>
</head>

<body>
  <script>
    var width = 800, height = 500, data=[];
    var URL = http://localhost/xampp/random.php;

    d3.text(URL, function(textLines) {
      textLines.forEach(function(line) {
        data.push(parseFloat(line));
      });
    })
  </script>
</body>
</html>

```

Listing 8.2 is straightforward: The `<script>` element specifies a location for the URL variable that is used to invoke a PHP script in the `D3 .text()` method. The next portion of code uses a `forEach` loop to iterate through the data values and populate the JavaScript `data` array with those data values.

Listing 8.3 displays the contents of the PHP script `random.php` that generates a set of random numbers that are used in Listing 8.2. This PHP script is located in the following directory (but you can place this script in a different location):

```
/Applications/XAMPP/xamppfiles/htdocs/xampp
```

### **LISTING 8.3** *random.php*

```

<?
for ($counter=0; $counter<10; $counter+=1) {
  echo(rand(10,100));
  echo "<br />";
}
?>

```

Listing 8.3 contains a very simple PHP loop that prints ten randomly generated numbers (each one is between 10 and 100) in a single column.

## **D3 WITH MYSQL DATA**

This section provides a PHP script `users.php` that extracts data from a MySQL database table called `users`, which contains the first and last names of a set of users. In addition, you will see the contents of an HTML Web page that invoke the PHP script to retrieve the list of user names. This example does not render anything in the HTML Web page because its purpose is to show you how to retrieve data from a PHP script and then use the `D3 .text()`

method to populate a JavaScript array with the list of users. Keep in mind that the PHP script provides the logic and the PHP statements for retrieving data from a MySQL table: the code assumes that you have an installation of MySQL along with a suitably named database containing a table of user names. Once again, this section uses a XAMPP installation on a Macbook Pro, but you can use a separate installation of MySQL if you prefer.

#### **LISTING 8.4** *PHPMySQLData1.html*

```
<!DOCTYPE HTML>
<html>
  <head>
    <meta charset="utf-8">
    <title>Data via MySQL</title>

    <script src="d3.min.js"></script>
  </head>

  <body>
    <script>
      var width = 800, height = 500, data=[];
      var URL = http://localhost/xampp/users.php;

      d3.text(URL, function(userLines) {
        userLines.forEach(function(user) {
          data.push(user);
        });
      });
    </script>
  </body>
</html>
```

Listing 8.4 is straightforward: The `<script>` element specifies a location for the URL variable that is used to invoke a PHP script in the `D3 .text()` method. The next portion of code uses a `forEach` loop to iterate through the data values and populate the JavaScript `data` array with those data values.

Listing 8.5 displays the contents of the PHP script `users.php` that retrieves the list of users in a database table. This PHP script is located in the following directory (but you can place this script in a different location):

```
/Applications/XAMPP/xamppfiles/htdocs/xampp
```

#### **LISTING 8.5** *users.php*

```
<?php
// load a mysql server configuration file:
include 'mysqlConfig.php';

@mysql_select_db($myDatabase) or
    die( "Unable to select database");

$user_query = "SELECT FNAME, LNAME from USERS";
```

```
// output the data as lines of text
$result = mysql_query(user_query);

$rows = array();
$data = "";

while(($row = mysql_fetch_row($result))) {
    $data[] = $row[0] . " " . $row[1];
}

echo json_encode($data);

mysql_close();
?>
```

Listing 8.5 performs some initialization and retrieves a set of users from a database table. Next, a simple loop appends each row to the JavaScript array `data`, after which the data is returned as JSON-formatted data (but you can use another data format that is suitable for your needs).

## D3 BAR CHARTS WITH A WEBSOCKET SERVER

In Chapter 3, you learned how to render a bar chart, and you just learned how to update a bar chart with data that was retrieved with an Ajax invocation. In this section, you will learn how to dynamically update and render a bar chart on a client browser with data that is periodically pushed from a WebSocket server.

The code sample in this section consists of three files: the HTML5 Web page `JQWSBarChart1.html` for rendering the bar chart, the Python script `echo_wsh.py` (using Python 2.7.1) that sends data to the client browser, and the shell script `run.sh` that launches the WebSocket server.

One point to keep in mind is that some other browsers may implement older versions of the WebSockets protocol. For up-to-date information, visit this Website: [caniuse.com/#feat=websockets](http://caniuse.com/#feat=websockets).

Listing 8.6 displays a portion of the contents of `JQWSBarChart1.html` that illustrates how to receive data from a WebSocket server and refresh a bar chart with that data in HTML5 Canvas. Note that the code for the bar chart and the CSS stylesheet `CSS32Background1.css` have been omitted for brevity, but the complete listings for both files are available on the CD.



### **LISTING 8.6** *JQWSBarChart1.html*

```
<!DOCTYPE HTML>
<html lang="en">
<head>
  <meta charset="utf-8" />
  <title>WebSockets and Bar Chart</title>
```

```

<link href="CSS32Background1.css"
      rel="stylesheet" type="text/css">
<script src="http://code.jquery.com/jquery-1.7.1.min.js">
</script>

<script>
var currentX      = 0;
var currentY      = 0;
var barCount      = 10;
var barWidth      = 40;
var barHeight     = 0;
var maxHeight     = 300;
var xAxisWidth    = (2*barCount+1)*barWidth/2;
var yAxisHeight   = maxHeight;
var labelY        = 0;
var indentY       = 5;
var shadowX       = 2;
var shadowY       = 2;
var axisFontSize  = 12;
var fontSize      = 16;
var leftBorder    = 50;
var topBorder     = 50;
var arrowWidth    = 10;
var arrowHeight   = 6;
var barHeights    = new Array(barCount);
var fillColors    = ['#f00', '#0f0', '#ff0', '#00f'];
var elem, context, gradient1;

var WSData = "", sendMsg = "Client-side Message", receiveMsg = "";
var theTimeout, pollingDelay = 2000;

function initialGraph() {
  var foundCanvas = setup();

  if(foundCanvas == true) {
    getWebSocketData();
  }
}

function setup() {
  // Get the canvas element
  elem = document.getElementById('myCanvas');
  if(!elem || !elem.getContext) {
    return false;
  }

  // Get the canvas 2d context
  context = elem.getContext('2d');
  if(!context) {
    return false;
  }

  return true;
}

```



```

///// bar chart code omitted for brevity

function getWebSocketData() {
  if ("WebSocket" in window) {
    console.log("Your Browser supports WebSockets");

    // Open a web socket
    var ws = new WebSocket("ws://localhost:9998/echo");

    // specify handlers for open/close/message/error
    ws.onopen = function() {
      // Web Socket is connected, send data using send()
      ws.send(sendMsg);
      console.log("Sending Message to server: "+sendMsg);
    };

    ws.onmessage = function (evt) {
      var receivedMsg = evt.data;
      console.log("Message from server: "+receivedMsg);
      setupBarChart(receivedMsg);
    };

    ws.onclose = function() {
      // websocket is closed.
      console.log("Connection is closed...");
    };

    ws.onerror = function(evt) {
      console.log("Error occurred: "+evt.data);
    };
  } else {
    // The browser doesn't support WebSocket
    console.log("WebSocket NOT supported by your Browser!");
  }
}

function setupBarChart(dataStr) {
  // populate the barHeights array
  barHeights = dataStr.split(" ");
  barCount    = barHeights.length;

  drawGraph2();
}
</script>
</head>

<body onload="initialGraph();">
  <script>
    $(document).ready(function() {
      $("#StartClientPoll").click(function() {
        getWebSocketData();
        setTimeout("getWebSocketData()", pollingDelay);
      })

      $("#StopClientPoll").click(function() {

```

```

        clearInterval(theTimeout);
        theTimeout = null;
    })

    $("#StartWSPull").click(function() {
        getWebSocketData();
        theTimeout = setInterval("getWebSocketData()", pollingDelay);
    })

    $("#StopWSPull").click(function() {
        clearInterval(theTimeout);
        theTimeout = null;
    })
});
</script>

<div id="conn"></div>

<div>
    <canvas id="myCanvas" width="800" height="400">No support for Canvas
        alt="Example rendering of a bar chart">
    </canvas>
</div>

<div id="WSDataDiv">
    <input type="button" id="StartClientPoll"
        value="Start Client Polling" />
    <input type="button" id="StopClientPoll"
        value="Stop Client Polling" />
    <input type="button" id="StartWSPull"
        value="Start Server" />
    <input type="button" id="StopWSPull"
        value="Stop Server" />
</div>
</body>
</html>

```

The HTML `<body>` element in Listing 8.6 invokes the function `initialGraph()` that performs some initialization and then invokes `getWebSocketData()` to get the bar-chart data (if the HTML5 `<canvas>` element exists in the Web page), as shown here:

```

function initialGraph() {
    var foundCanvas = setup();

    if(foundCanvas == true) {
        getWebSocketData();
    }
}

```

Next, the function `getWebSocketData()` implements the standard callback functions in JavaScript, including the `onmessage` function, which

gets the data from the WebSocket server and passes the data to the `setupBarChart()` function for rendering the bar chart, as shown here:

```
function getWebSocketData() {
  if ("WebSocket" in window) {
    // Open a web socket
    var ws = new WebSocket("ws://localhost:9998/echo");

    // code omitted for brevity
    ws.onmessage = function (evt) {
      var receivedMsg = evt.data;
      console.log("Message from server: "+receivedMsg);
      setupBarChart(receivedMsg);
    };
  }
}
```

As you know, the code samples in this book render correctly in WebKit-based browsers, and in some cases the code samples work in other browsers. In particular, Firefox 11 and beyond supports the code in Listing 8.6.

The `setupBarChart()` JavaScript function accepts a string parameter that consists of a space-delimited set of values for the bar heights of the bar chart, which is used to populate a JavaScript array with those bar heights, as shown here:

```
function setupBarChart(dataStr) {
  // populate the barHeights array
  barHeights = dataStr.split(" ");

  barCount    = barHeights.length;

  drawGraph2();
}
```

The rendering of the bar chart is handled by the JavaScript function `drawGraph2()`, which is not shown in Listing 8.6.

The other part to consider is the code on the WebSocket server that returns the data values for the bar chart. The data is retrieved with the following code snippet that is contained in the `getWebSocketData()` function, as shown here:

```
// Open a web socket
var ws = new WebSocket("ws://localhost:9998/echo");
```

Now let's turn our attention to the server-side code. Before delving into the details of the code, you need to download `pywebsocket`, which can be found at <https://code.google.com/p/pywebsocket/>. Let's examine the relevant portion of the Python script `echo_wsh.py` that sends data from the WebSockets server to the client (via port 9998), whose contents are displayed in Listing 8.7.

**LISTING 8.7** *echo\_wsh.py*

```

# Copyright 2011, Google Inc.
# All rights reserved.
#
# Redistribution and use in source and binary forms, with or without
# modification, are permitted provided that the following conditions are
# met:
#
#     * Redistributions of source code must retain the above copyright
# notice, this list of conditions and the following disclaimer.
#     * Redistributions in binary form must reproduce the above
# copyright notice, this list of conditions and the following disclaimer
# in the documentation and/or other materials provided with the
# distribution.
#     * Neither the name of Google Inc. nor the names of its
# contributors may be used to endorse or promote products derived from
# this software without specific prior written permission.
#
# THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
# "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
# LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
# A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
# OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
# SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
# LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
# DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
# THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
# (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
# OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

_GOODBYE_MESSAGE = u'Goodbye'

from random import randint

def web_socket_do_extra_handshake(request):
    pass # Always accept.

def web_socket_transfer_data(request):
    while True:
        line = request.ws_stream.receive_message()
        if line is None:
            return
        if isinstance(line, unicode):
            barCount = 10;
            randomValues = ""

            for x in range(1, barCount):
                randomValues = randomValues + str(randint(50, 300))+ " "

            print 'randomValues2: %s' % randomValues
            request.ws_stream.send_message(randomValues, binary=False)

        if line == _GOODBYE_MESSAGE:
            return
        else:
            print 'Sending2: %s' % line
            request.ws_stream.send_message(line, binary=True)

```

The code shown in bold in Listing 8.7 is the code that creates a string containing a randomly generated set of integers that represent the ten bar heights of a bar chart:

```
barCount = 10;
randomValues = ""

for x in range(1, barCount):
    randomValues = randomValues + str(randint(50, 300))+" "

    print 'randomValues2: %s' % randomValues
```

After the string of concatenated bar heights has been created, it is sent to the client with the following code snippet:

```
request.ws_stream.send_message(randomValues, binary=False)
```

The third file that we need is a simple Bourne shell script `run.sh` that launches the WebSocket server on port 9998.

#### **LISTING 8.8** *run.sh*

```
CURRDIR=`pwd`
PYTHONPATH=$CURRDIR
python ./mod_pywebsocket/standalone.py -p 9998 -d $CURRDIR/example
```

Listing 8.8 is straightforward: it sets two shell-script variables and then launches the Python script `standalone.py` located in the `mod_pywebsocket` subdirectory, specifying port 9998 and the `example` subdirectory of `$CURRDIR`, which is the parent directory of the Python script `echo_wsh.py` in Listing 8.8.

Open a command shell, navigate to the subdirectory `pywebsocket/src`, and launch the shell script in Listing 8.8 as follows:

```
./run.sh
```

Listing 8.8 contains a copyright notice that is required for displaying the contents of this Python script. The new section of code (displayed in bold in Listing 8.8), is shown here:

```
barCount = 10;
randomValues = ""

for x in range(1, barCount):
    randomValues = randomValues + str(randint(50, 300))+" "

request.ws_stream.send_message(randomValues, binary=False)
```

Figure 8.1 displays a very colorful bar chart and the data that is returned from a server, which specifies the height of each bar element in the graph.



FIGURE 8.1 A Bar Chart Using HTML5 WebSockets.

If you prefer using open-source toolkits for handling the various WebSocket details for multiple browsers there are also jQuery plugins available that use jQuery as a layer of abstraction on top of WebSockets (with support for multiple browsers), such as the one that is here:

<https://code.google.com/p/jquery-graceful-websocket/>

This jQuery plugin provides fallback support for Ajax if WebSockets is unavailable along with options to override the default options.

## D3 AND NODEJS (OPTIONAL)

This optional section discusses D3 in the context of NodeJS; you can skip this section without any loss of continuity. If you are interested in NodeJS, you can find plenty of free online tutorials that explain the principles of NodeJS along with download links and sample programs.

If you have NodeJS already installed on your machine, you can install D3 as a NodeJS module with the following command:

```
npm install -g d3
```

### Inserting an `<svg>` Element in an HTML Web Page

Listing 8.9 displays the contents of `SimpleSVGNode.js` that generate an HTML Web page with an `<svg>` element inserted as a child of an HTML `<div>` element, which in turn is a child of the `<body>` element.

**LISTING 8.9 SimpleSVGNode.js**

```

var d3 = require("d3");
var width=800, height=500;

d3.select("body").append("div")
  .attr("id", "chart")
  .append("svg")
  .attr("id", "background")
  .attr("width", 800)
  .attr("height", 500);

var htmlPage = window.document.innerHTML;

console.log(htmlPage);

```

Listing 8.9 starts with a `require("d3")` statement followed by the definition of two JavaScript variables. When you `require("d3")`, it creates a window and a document and automatically includes JSDOM.

Make sure that you add the line `<!DOCTYPE html>` because JSDOM does not have any knowledge of SVG. Although JSDOM can create SVG elements, it doesn't support SVG's special APIs (e.g., `getBBox`). Depending on what you are trying to do, you might have better luck with PhantomJS.

The next part of Listing 8.9 invokes the D3 `select` method to append a new HTML `<div>` element to the `<body>` element. The code uses so-called method chaining to invoke the D3 `select` method a second time, which inserts an `<svg>` element as a child of the previously created `<div>` element. Now launch the JavaScript file in Listing 8.9 as follows:

```
node SimpleSVGNode.js
```

Unfortunately, there is now a compatibility bug in NodeJS with `d3.min.js`, and when you invoke the preceding command, the following error message is displayed on the command line:

```

var d3_mouse_bug44083 = /WebKit/.test(navigator.userAgent) ? -1 : 0;
                                     ^
TypeError: Cannot read property 'userAgent' of undefined
    at /usr/local/lib/node_modules/d3/d3.v2.js:4104:50
    at Object.<anonymous>
    (/usr/local/lib/node_modules/d3/d3.v2.js:7026:3)
    at Module._compile (module.js:456:26)
    at Object.Module._extensions..js (module.js:474:10)
    at Module.load (module.js:356:32)
    at Function.Module._load (module.js:312:12)
    at Module.require (module.js:364:17)
    at require (module.js:380:17)
    at Object.<anonymous>
    (/usr/local/lib/node_modules/d3/index.js:16:1)
    at Module._compile (module.js:456:26)

```

Two other online posts contain the same error (under different circumstances), and because this code launched successfully with earlier versions of

NodeJS and d3.js, it's possible that this will be resolved in future code releases. The inclusion of this code sample and the code sample in the next section is to give you an idea of how you can use D3 with NodeJS.

Listing 8.10 displays the contents of `SimpleSVGNode.html` that is generated when you invoke the preceding node command.

**LISTING 8.10 SimpleSVGNode.html**

```
<html>
<head></head>
<body>
  <div id="chart">
    <svg id="background" width="800" height="500"></svg>
  </div>
</body>
</html>
```

As you can see, Listing 8.10 is an HTML Web page with a `<div>` element that contains an empty SVG `<svg>` element.

## Rendering SVG Graphics with D3 and NodeJS (Optional)

The code sample in this section requires some knowledge of NodeJS and express, but it's possible to understand the basic purpose even if you are new to these technologies.

Listing 8.11 displays the contents of `ArchEllipses1.js` that generates a set of Archimedean-based ellipses in an HTML Web page.

**LISTING 8.11 ArchEllipses1.js**

```
var d3 = require("d3");

var express = require("express");
var app = express();

var basePointX    = 200,
    basePointY    = 220,
    offsetX       = 0,
    offsetY       = 0,
    minorAxis     = 80,
    majorAxis     = 50,
    Constant      = 0.25,
    maxAngle      = 720;

var w = 600, h = 400, p = 30,
    eColors = ['#f00', '#0f0', '#00f'],
    x = d3.scale.linear().domain([0, 1]).range([0, w]),
    y = d3.scale.linear().domain([0, 1]).range([h, 0]);

app.get('/', function(req,res){
  var data = d3.range(maxAngle).map(function(a) {
    return {angle:  a,
            radius:  Constant*a,
            offsetX: Constant*a*Math.cos(a*Math.PI/180),
```



```

        offsetY: Constant*a*Math.sin(a*Math.PI/180)}
    });

    var vis = d3.select("body")
        .append("svg:svg")
        .data([data])
        .attr("width", w + p * 2)
        .attr("height", h + p * 2)
        .append("svg:g")
        .attr("transform", "translate(" + p + "," + p + ")");

    vis.selectAll("text")
        .data(data)
        .enter().append("svg:ellipse")
        .attr("fill", function(d) {
            return eColors[d.angle % eColors.length]; })
        .attr("cx", function(d) { return basePointX+d.offsetX; })
        .attr("cy", function(d) { return basePointY+d.offsetY; })
        .attr("rx", function(d) { return majorAxis; })
        .attr("ry", function(d) { return minorAxis; });

    var htmlPage = window.document.innerHTML;
    res.send(htmlPage);
});

app.listen(3003);
console.log("listening on port 3003");

```

Listing 8.11 starts by initializing an express-based application and then initializes some JavaScript variables. When users navigate to `http://localhost:3003` in a Web browser, a GET request is issued, which is handled by the section of code in Listing 8.11 that starts with this code snippet:

```

app.get('/', function(req,res){
    // do something here
})

```

In our case, the preceding code block contains a loop that uses D3 to generate a set of ellipses that follow the path of an Archimedean spiral (the details of which are already familiar to you). In this case, there is a `vis` variable that first creates an `<svg>` element and uses method chaining to set various attributes of this `<svg>` element. Next, the `vis` variable creates a set of SVG `<ellipse>` elements (and sets the attributes of each ellipse) and appends each `<ellipse>` element to the `<svg>` element via an implicit loop.

When the loop finishes execution, the contents of the HTML Web page are retrieved and sent back to the browser that initiated the request using these two lines of code:

```

var htmlPage = window.document.innerHTML;
res.send(htmlPage);

```

Launch the code in Listing 8.11 with the following command:

```
node ArchEllipses1.js
```

An open-source project with many similar D3-based code samples (but without using NodeJS) is available at <http://code.google.com/p/d3-graphics>.

## ADDITIONAL CODE SAMPLES ON THE CD

---



The HTML Web page `BarChartWeatherDataBlurFilter1.html` is the counterpart to the HTML Web page `BarChartWeatherData.html`, augmented by a Gaussian blur filter.

The HTML Web page `BarChartWeatherDataGradient1.html` is the counterpart to the HTML Web page `BarChartWeatherData.html`, augmented with a linear gradient and a radial gradient.

## SUMMARY

---

This chapter started with an example of using D3 with Ajax. Next, you saw code samples that combine D3 with PHP followed by an example of using D3 with data that is stored in a MySQL database. Finally, you saw an example of accessing and then rendering live data using an HTML5 WebSocket server and a NodeJS server. The next chapter shows you how to use D3 with other JavaScript toolkits.



# MISCELLANEOUS D3 APPLICATION PROGRAMMING INTERFACES AND OTHER TOOLKITS



This chapter contains code samples for an assortment of D3 APIs such as maps, layouts, trees, Voronoi diagrams, as well as D3 plugins and extensions. In addition, the CD contains appendices that introduce you to several interesting JavaScript toolkits, and they provide you with a starting point for deciding whether or not to use these toolkits in conjunction with D3 in your HTML Web pages.

The first part of this chapter provides an example of rendering a map of the United States with various colors and how to use the D3 Force API for creating graphs in which nodes can interact with each other (via attraction and repulsion). The second part of this chapter shows you how to render a tree in D3 and how to use the D3 API for rendering a Voronoi diagram from CSV data. The final part of this chapter discusses a D3 plugin called Cubic (developed by Mike Bostock) as well as extensions of D3 (such as Rickshaw and dc.js).

Please keep in mind that the examples in this chapter are intentionally simplified because this is an introductory book about D3, so the code samples focus on conveying the concepts that underlie the APIs. After reading this chapter, you can find more advanced code examples by performing an Internet search. In the event that those code samples do not meet your needs, you can post your questions (along with your work-in-progress code) on the D3 online forum.

## MAPS IN D3 (CHOROPLETH)

---

According to Wikipedia, a choropleth map “is a thematic map in which areas are shaded or patterned in proportion to the measurement of the statistical variable being displayed on the map, such as population density or per-capita income.” Two examples are available here:

<http://bl.ocks.org/3306362>

<https://secure.polisci.ohio-state.edu/faq/d3/afghanistan.html>



The HTML Web page `USMapColorsAndFilter1.html` is 364 lines long, and only 25 of those lines are actual code (the rest is mostly data values). Listing 9.1 displays only the contents of the `<script>` element in this HTML Web page, but the entire code listing is available on the CD. The code block in Listing 9.1 illustrates how to change the colors and apply an SVG filter to a blank United States map.

### **LISTING 9.1** *JavaScript for the Map*

```
<script>
  var index=0, fillColors=["red", "green", "#cc0", "blue"];
  var strokeColor="black", strokeWidth="4", stdDev=2;
  var scaleX=0.6, scaleY=0.6;

  var svg = d3.select("svg");

  var filter = svg.append("svg:defs")
    .append("svg:filter")
    .attr("id", "blurFilter1")
    .append("svg:feGaussianBlur")
    .attr("stdDeviation", stdDev);

  d3.selectAll("path")
    .attr("fill", function(d, i) {
      index = i % fillColors.length;
      return fillColors[index];
    })
    .attr("filter", "url(#blurFilter1)")
    .attr("transform", "scale("+scaleX+", "+scaleY+")")
</script>
```

Listing 9.1 initializes some JavaScript variables and then defines the JavaScript variable `filter` that specifies a Gaussian blur filter in D3 (which you first saw in Chapter 4). The next portion of Listing 9.1 references the existing SVG `<path>` elements (not shown in Listing 9.1) whose data points represent each state of the United States. The color assignment for each state is handled in the following code block:

```
.attr("fill", function(d, i) {
  index = i % fillColors.length;
  return fillColors[index];
})
```

In addition, the Gaussian blur filter is applied to each state along with a scaling effect, as shown in this code snippet:

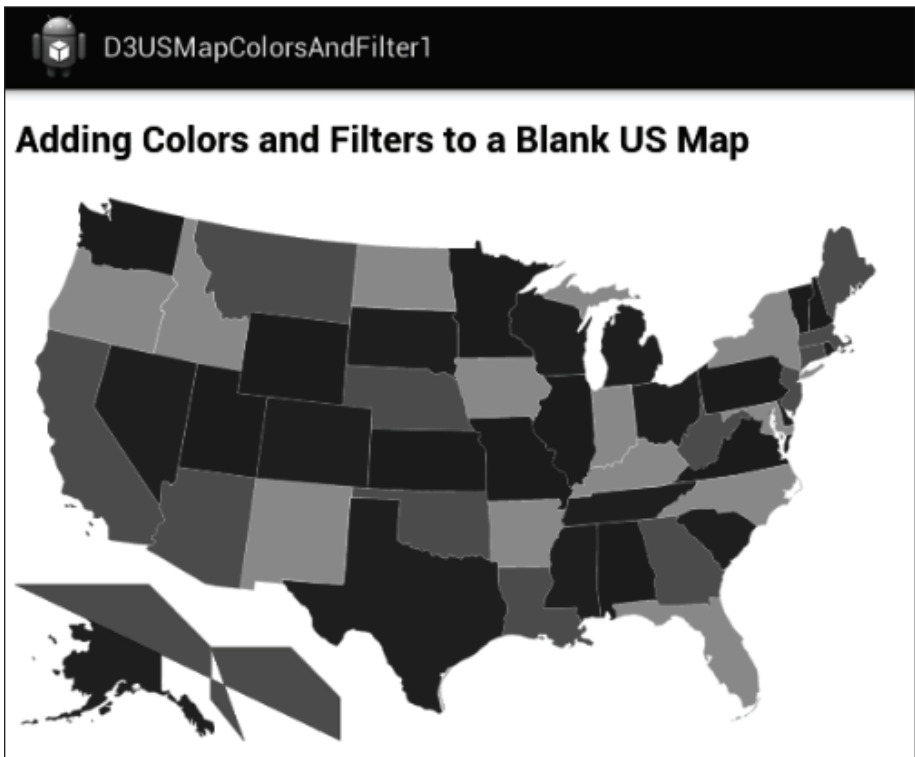
```
.attr("filter", "url(#blurFilter1)")
.attr("transform", "scale("+scaleX+", "+scaleY+")")
```

In case you are curious, each SVG `<path>` element representing a state also contains an `id` attribute that specifies the two-letter abbreviation for that

state. For example, the two-character string `NH` is the abbreviation for the state of New Hampshire, and the following SVG `<path>` element specifies the data points that represent the state of New Hampshire:

```
<path
style="fill-opacity:1;stroke:#ffffff;stroke-opacity:1;stroke-
width:0.75;stroke-miterlimit:4;stroke-dasharray:none"
id="NH"
d="M 880.79902,142.42476 L 881.66802,141.34826 L 882.75824,138.05724
L 880.21516,137.14377 L 879.73017,134.07221 L 875.85032,132.94059
L 875.527,130.19235 L 868.25225,106.75153 L 863.65083,92.208542 L
862.75375,92.203482 L 862.10711,93.820087 L 861.46047,93.335106 L
860.4905,92.365143 L 859.03556,94.305068 L 858.98709,99.337122 L
859.29874,105.00434 L 861.23866,107.75258 L 861.23866,111.7941 L
857.52046,116.85688 L 854.93389,117.98852 L 854.93389,119.12014 L
856.06552,120.89841 L 856.06552,129.46643 L 855.25721,138.6811 L
855.09555,143.53092 L 856.06552,144.82422 L 855.90386,149.35071 L
855.41887,151.12899 L 856.38768,151.83821 L 873.17535,147.41366 L
875.35022,146.81121 L 877.19379,144.03788 L 880.79902,142.42476
z" />
```

Figure 9.1 displays the map of the United States with the coloring that is defined in Listing 9.1. Although the map and the colors for each of the states



**FIGURE 9.1** Coloring a Map of the United States.

are rendered correctly, the SVG `<filter>` effect is not displayed on an Asus Prime 10-inch tablet with Android ICS.

## ADDING TOOLTIPS TO A UNITED STATES MAP

Now that you know how to render a blank map of the United States, you can color the individual states based on different sets of data, such as census data, average state rainfall, average monthly temperature, and so forth.



The HTML Web page `USMapAnnualRainfall.html` also contains mostly data, and Listing 9.2 displays the portion of code that is relevant to our discussion (but the full source listing is available on the CD). In fact, Listing 9.2 displays only the contents of the `<script>` element that contain code logic that you can use for displaying a gradient-colored map with the average yearly rainfall for each state. When users hover over any state, the rainfall for that state is displayed as a tooltip.

In this code sample, the rainfall values are generated randomly, but obviously in a real scenario you would replace these random values with actual values (which you can find in an online almanac).

### LISTING 9.2 *JavaScript for Rainfall*

```
<script>
  var stateCodes = [
    "AL", "AK", "AZ", "AR", "CA", "CO", "CT", "DE", "FL", "GA",
    "HI", "ID", "IL", "IN", "IA", "KS", "KY", "LA", "ME", "MD",
    "MA", "MI", "MN", "MS", "MO", "MT", "NE", "NV", "NH", "NJ",
    "NM", "NY", "NC", "ND", "OH", "OK", "OR", "PA", "RI", "SC",
    "SD", "TN", "TX", "UT", "VT", "VA", "WA", "WV", "WI", "WY"
  ];

  var scaleX=0.6, scaleY=0.6;
  var rVal=0, gVal=0, bVal=0, index=0, color="";
  var randomAmount=0, maxRainFall=0, stateRain=[], stateColors=[];
  var id="", staterain=0, stateIDRainfall=[];

  for(var x=0; x<stateCodes.length; x++) {
    randomAmount = 255*Math.random();
    stateRain.push(randomAmount);
  }

  var maxRainFall = d3.max(stateRain);

  for(var x=0; x<stateRain.length; x++) {
    rVal = Math.floor(255*stateRain[x]/maxRainFall);
    color = "rgb("+rVal+",0,0)";
    stateColors.push(color);
    stateIDRainfall[stateCodes[x]] = rVal;
  }

  var svg = d3.select("svg");
```

```

d3.selectAll("path")
  .attr("fill", function(d, i) {
    index = i % stateRain.length;
    return stateColors[index];
  })
  .attr("transform", "scale("+scaleX+", "+scaleY+")")
  .append("svg:title")
    .text(function() {
      var parent = d3.select(this).node().parentNode;
      id = d3.select(parent).attr("id");
      staterain = stateIDRainfall[id];
      return staterain;
    })
</script>

```

Listing 9.2 initializes some JavaScript variables followed by a loop that populates an array of color values for the states of the United States. Each color value is based a scaled value for the red component of the (R,G,B) color value, based on its relative amount of rainfall, as shown here:

```

for(var x=0; x<stateRain.length; x++) {
  rVal = Math.floor(255*stateRain[x]/maxRainFall);
  color = "rgb("+rVal+",0,0)";
  stateColors.push(color);
  stateIDRainfall[stateCodes[x]] = rVal;
}

```

As you can see, states with heavier rainfall are brighter red, and states with the lowest rainfall are closer to black (this isn't necessarily a realistic way to render colors, but you get the idea).

The next portion of code in Listing 9.2 assigns the calculated colors to each state in the map and scales the size of the map (it's reduced by 50% in this code sample). The final portion of code adds tooltip functionality, so the rainfall for a state is displayed when users hover over that state with their mouse, as shown here:

```

.append("svg:title")
  .text(function() {
    var parent = d3.select(this).node().parentNode;
    id = d3.select(parent).attr("id");
    staterain = stateIDRainfall[id];
    return staterain;
  })

```

Notice how the preceding code snippet first selects the parent of the current node (which is the state where users are hovering with their mouse) and then finds the value of its `id` attribute. Next, the code uses the value of the `id` attribute as an index into the `stateIDRainfall` array (which was populated in the first part of Listing 9.2) to find and then return this value.



## D3 AND GOOGLE MAPS

The preceding code samples showed you how to use D3 to manipulate maps with SVG-based data. As you might have already discovered, you can also use other toolkits in conjunction with D3 to render maps. For example, you can easily create maps using D3 combined with Google Maps, whose latest release is version 3 (and version 2 is now deprecated). However, keep in mind that these toolkits do change their APIs, so you might need to revise code samples (such as the one mentioned in this section) that use APIs from Google Maps or other third-party toolkits. A good introductory example that shows you how to use D3 with Google Maps can be found at <https://gist.github.com/mbostock/899711>.

There are a few points to notice in the code in the preceding Website. There is an HTML `<div>` element whose `id` attribute has the value `map` followed by the JavaScript code that references this `<div>` element during the map initialization, as shown here:

```
<body>
<div id="map"></div>
<script>
  // Create the Google Map...
  var map = new google.maps.Map(d3.select("#map").node(), {
    zoom: 8,
    center: new google.maps.LatLng(37.76487, -122.41948),
    mapTypeId: google.maps.MapTypeId.TERRAIN
  });
```

The preceding code block is a standard way of initializing a Google Map (currently available in version 3). The next portion of code uses the `D3.json()` method to retrieve the map data, as shown here:

```
<div id="map"></div>
d3.json("stations.json", function(data) {
  var overlay = new google.maps.OverlayView();
  // various D3 snippets for markers and labels,
  // using D3 code that is very familiar to you
})
```

The omitted D3 code snippets are straightforward: they start with the `TMCHID3` followed by the creation of an SVG `<circle>` element an SVG `<text>` element.

Note that the contents of the JSON-based data in the file `stations.json` are available online at the Website mentioned at the beginning of this section.

## GEOJSON AND D3 TOPOJSON

GeoJSON is a JSON-based open format for representing geographic data in a human readable form. GeoJSON supports various data types including

points, polygons, multipolygons, features, geometry collections, and bounding boxes, which are stored along with feature information and attributes. Moreover, because GeoJSON data is also JSON data, you can use JSON tools for manipulating GeoJSON data.

TopoJSON is an extension of GeoJSON that provides several advantages over GeoJSON, as described by Mike Bostock and reproduced here:

TopoJSON is an extension of GeoJSON that encodes topology. Rather than representing geometries discretely, geometries in TopoJSON files are stitched together from shared line segments called arcs. TopoJSON eliminates redundancy, offering much more compact representations of geometry than with GeoJSON; typical TopoJSON files are 80% smaller than their GeoJSON equivalents. In addition, TopoJSON facilitates applications that use topology, such as topology-preserving shape simplification, automatic map coloring, and cartograms.

D3 supports TopoJSON (and therefore GeoJSON), which you can download at <https://github.com/mbostock/topojson>. You can install TopoJSON on your machine with this command:

```
npm install -g topojson
```

If you want to create a map in TopoJSON, follow the instructions in Mike Bostock's tutorial, found at <http://bost.ocks.org/mike/map/>. Please keep in mind that this link requires the FWTools open-source toolkit (currently available only for Windows and Linux), found at <http://fwtools.maptools.org/>.

## OTHER MAPS

---

The preceding sections have shown you how to use D3 to manipulate maps consisting of SVG-based data. However, you can also use the `D3.json()` method with maps that are stored in CartoDB, which supports SQL queries for retrieving map-related data.

Other types of maps are also available. For example, a height map is a map that resembles a heat map that measures the altitude above sea level of various locations. A vivid example can be found at <http://bl.ocks.org/mbostock/3289530>. The interesting point about this example is that it shows you how to remap a monochrome PNG image using a custom color scale in D3 (which is not covered in this chapter).

An example of a heat map that uses JSON and HTML5 Canvas can be found at <http://bl.ocks.org/mbostock/3074470>.

The following example displays the airports in the United States, and whenever you hover over an airport (represented by a circle) you will see the links to other airports:

*<http://mbostock.github.com/d3/talk/20111116/airports.html>.*

## THE D3 FORCE LAYOUT

D3 provides APIs for various layouts including bundle, chord, clusters, and force. A useful Website that contains links with details regarding the twelve D3 layout types is located at <https://github.com/mbostock/d3/wiki/Layouts>.

A D3 force layout is one of the D3 layouts, and it enables you to position nodes via a physical simulation. Consequently, a D3 force layout involves animation effects for positioning nodes based on their level of repulsion or attraction with each other (which you can specify programmatically).

Instead of providing a detailed description of the various parameters (and their effects) in a force-directed layout, it's worth your time to watch the following short video of Mike Bostock discussing the force API in D3: <http://vimeo.com/29458354>.

Listing 9.3 displays the contents of `Force1.html` that illustrate how to use the D3 force layout.

### LISTING 9.3 *Force1.html*

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>D3 Force Example</title>
    <script src="d3.min.js"></script>
  </head>

  <body>
    <script>
      var width = 500, height = 300;

      // original data
      var dataValues = {
        nodes: [
          { name: "Pizza" },
          { name: "Beer" },
          { name: "Wine" },
          { name: "Champagne" },
          { name: "Pizza" },
          { name: "Beer" },
          { name: "Wine" },
          { name: "Champagne" }
        ],
        edges: [
          { source: 0, target: 1 },
          { source: 1, target: 2 },
          { source: 2, target: 3 },
          { source: 3, target: 4 },
          { source: 0, target: 1 },
          { source: 1, target: 2 },
          { source: 2, target: 3 },
          { source: 3, target: 4 }
        ]
      };
    </script>
  </body>
</html>
```

```

// Initialize a default force directed layout
var force = d3.layout.force()
    .nodes(dataValues.nodes)
    .links(dataValues.edges)
    .size([width, height])
    .linkDistance([50])
    .charge([-100])
    .start();

var colors = d3.scale.category10();

//Create SVG element
var svg = d3.select("body")
    .append("svg")
    .attr("width", width)
    .attr("height", height);

//Create edges as lines
var edges = svg.selectAll("line")
    .data(dataValues.edges)
    .enter()
    .append("line")
    .style("stroke", "#ccc")
    .style("stroke-width", 1);

//Create nodes as circles
var nodes = svg.selectAll("circle")
    .data(dataValues.nodes)
    .enter()
    .append("circle")
    .attr("r", 10)
    .style("fill", function(d, i) {
        return colors(i);
    })
    .call(force.drag);

// This is invoked after every simulation "tick"
force.on("tick", function() {
    edges.attr("x1", function(d) { return d.source.x; })
        .attr("y1", function(d) { return d.source.y; })
        .attr("x2", function(d) { return d.target.x; })
        .attr("y2", function(d) { return d.target.y; });

    nodes.attr("cx", function(d) { return d.x; })
        .attr("cy", function(d) { return d.y; });
});
</script>
</body>
</html>

```

Listing 9.3 contains boilerplate code and a `<script>` element that initializes some JavaScript variables including the JavaScript `dataValues` array that specifies nodes and edges for the force diagram.

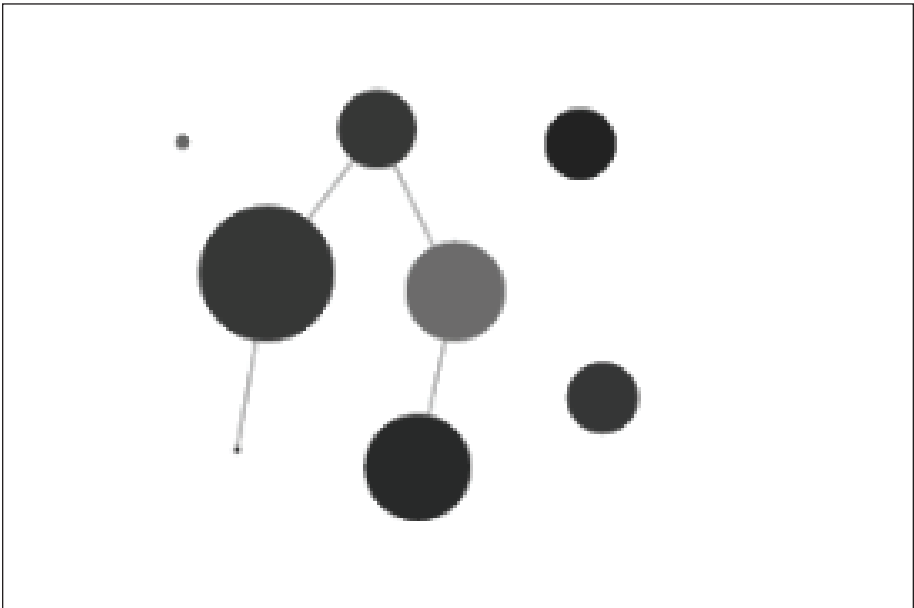
The heart of the D3 force code initializes a set of attributes and then invokes the `D3.start()` method, as shown here:

```
var force = d3.layout.force()
    .nodes(dataValues.nodes)
    .links(dataValues.edges)
    .size([width, height])
    .linkDistance([50])
    .charge([-100])
    .start();
```

The `nodes` attribute and the `links` attribute specifies the corresponding named components of the `dataValues` array. Note that nodes are typically mapped to SVG `<circle>` elements, and links are typically mapped to SVG `<line>` elements. The `size` attribute specifies two numbers that determine the gravitational center and the initial random position of each node. Negative values for the `charge` attribute produce repulsion between nodes, whereas positive values produce attraction. The `target` attribute sets the distance between nodes.

The preceding description gives a very brief overview of the purpose of the attributes that are specified in the preceding code snippet. A fuller description of these (and other) `force` attributes is found at <https://github.com/mbostock/d3/wiki/Force-Layout>.

Figure 9.2 displays a force diagram in D3 that is defined in Listing 9.3.



**FIGURE 9.2** Rendering a Force Diagram in D3.



```

-webkit-repeating-linear-gradient(-45deg, red 5px,
                                   green 4px, yellow 8px, blue 12px,
                                   transparent 16px, red 20px, blue 24px,
                                   transparent 28px, transparent 32px),
-webkit-radial-gradient(blue 8px, transparent 68px);

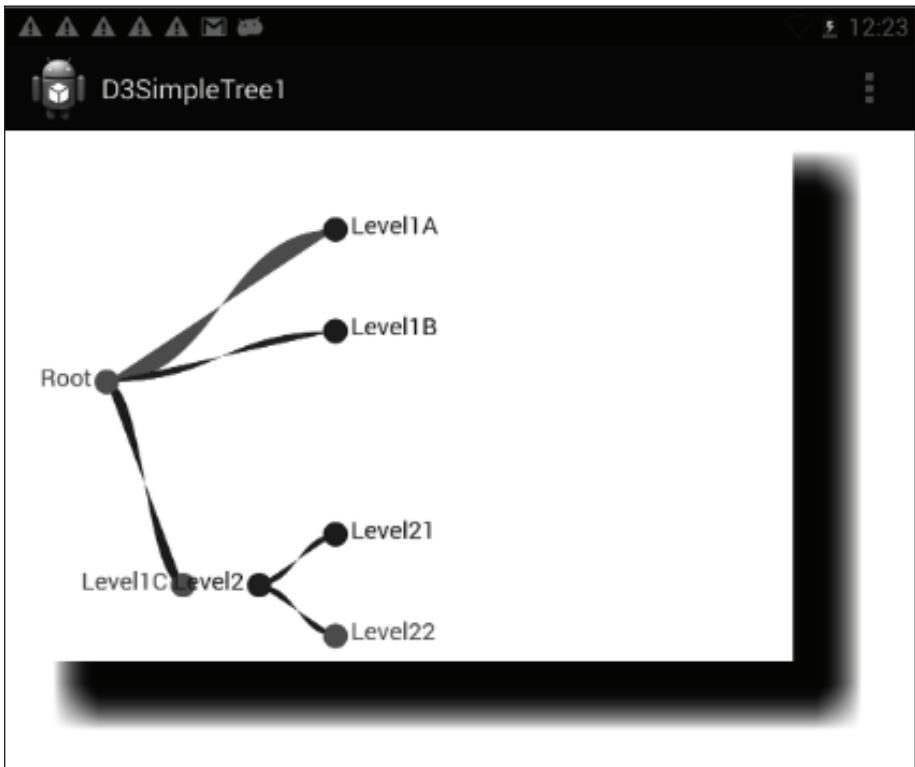
background-size: 80px 80px, 4px 4px;
background-position: 0 0;
}

```

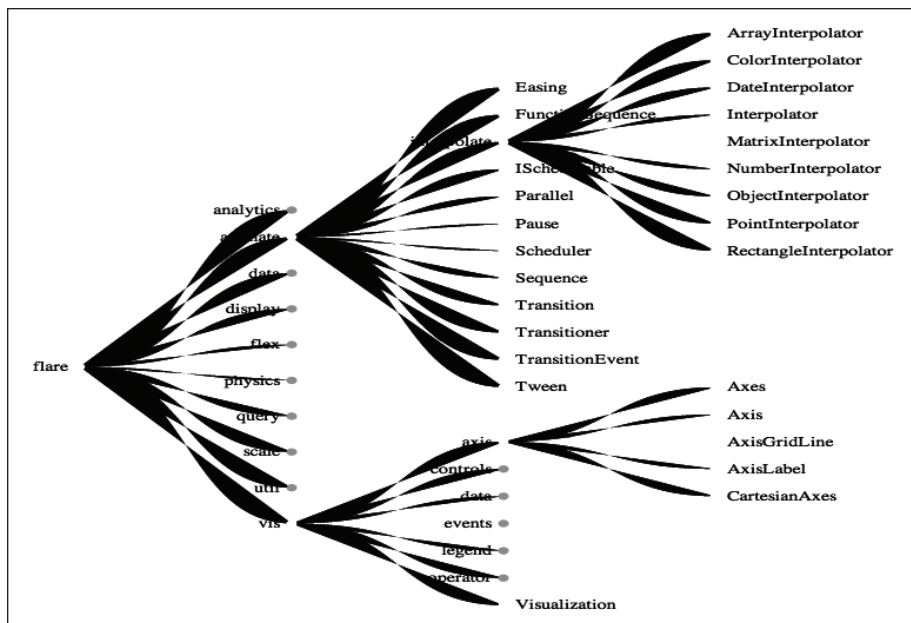
Listing 9.4 contains a CSS selector that matches the SVG `<svg>` element, and this selector contains properties that you saw in the CSS3 chapter. The second CSS selector matches the HTML `<div>` element (whose `id` attribute has the value `tree`) during a hover event.

Figure 9.3 displays a tree created with the D3 code in Listing 9.4. Note that the CSS3-based effects are not fully displayed on an Asus Prime 10-inch tablet with Android ICS (Ice Cream Sandwich).

A nice example of a D3-based tree with collapsible and expandable nodes can be found at <http://mbostock.github.com/d3/talk/20111018/tree.html>. This HTML Web page contains an SVG document that consists of a set of SVG `<text>` elements for displaying the text associated with each node in the tree.



**FIGURE 9.3** Rendering a Tree-like Data Set in D3.



**FIGURE 9.4** An SVG-based Tree-like Structure.

The arc between any pair of linked nodes is an SVG `<path>` element. A typical SVG `<path>` element is shown here:

```
<path class="link" d="M360,528.695652173913C450,528.695652173913
450,462.60869565217394 540,462.60869565217394"></path>
```



The CD contains the SVG document `MikeBostock-20111018-tree.svg` that is the SVG data in the online tree-based D3 code sample.

Figure 9.4 displays a tree created from `MikeBostock-20111018-tree.svg`, which is clearly different from the online version. This difference underscores the importance of using CSS selectors to style the elements in an HTML Web page.

## VORONOI DIAGRAMS

Wikipedia provides the following description of a Voronoi diagram:

In mathematics, a Voronoi diagram is a way of dividing space into a number of regions. A set of points (called seeds, sites, or generators) is specified beforehand, and for each seed there will be a corresponding region consisting of all points closer to that seed than to any other. The regions are called Voronoi cells. It is dual to the Delaunay triangulation. It is named after Georgy Voronoi and is also called a Voronoi tessellation, a Voronoi decomposition, or a Dirichlet tessellation (after Lejeune Dirichlet).



You can see an example of a Voronoi diagram (which might be more useful than the preceding definition) on the Wikipedia link found at [http://en.wikipedia.org/wiki/Voronoi\\_diagram](http://en.wikipedia.org/wiki/Voronoi_diagram).

The code in this section is based on the following code sample:

*<http://stackoverflow.com/questions/13091506/drawing-voronoi-diagram-from-a-csv-file-with-d3-js>*

Listing 9.5 displays the contents of `CSVVoronoi1.html` that illustrate how to use CSV data in a CSV file to render a Voronoi diagram.

#### **LISTING 9.5 CSVVoronoi1.html**

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8" />
    <title>CSV elements for Voronoi</title>
    <script src="d3.min.js"></script>
  </head>

  <body>
    <div id="chart"></div>

    <script>
      var width=900, height=500, index=0, radius=2;
      var fontSize="80%", textX=150, textY=200;
      var textStroke="orange", textColor="orange";
      var fillColors = ["red", "yellow", "blue", "green"];
      var textStr="control", circleFill="white";
      var hoverColor="black", hoverRadius=4*radius;
      var hoverStroke="white", hoverStrokeWidth=3;

      // GET THE DATA FROM THE CSV FILE AND PARSE IT:
      d3.text("Voronoi.csv", function(datasetText) {
        var vertices = d3.csv.parseRows(
          datasetText,
          function(pt) {
            return [parseFloat(pt[0]),parseFloat(pt[1])];
          }
        )

        var svg = d3.select("#chart")
          .append("svg")
          .attr("width", width)
          .attr("height", height)

        svg.selectAll("path")
          .data(d3.geom.voronoi(vertices))
          .enter().append("path")
          .attr("d", function(d) {
            return "M" + d.join("L") + "Z";
          })
          .attr("fill", function(d, i) {
            index = i % fillColors.length;
```

```

        return fillColors[index];
    })

    svg.selectAll("circle")
      .data(vertices.slice(1))
      .enter().append("circle")
      .attr("transform", function(d) {
        return "translate("+d+")";
      })
      .attr("r", radius)
      .attr("fill", circleFill)
      .on("mouseover", function() {
        d3.select(this)
          .attr("fill", hoverColor)
          .attr("stroke", hoverStroke)
          .attr("stroke-width", hoverStrokeWidth)
          .attr("r", hoverRadius)
      })

    text1 = svg.append("svg:text")
      .text(textStr)
      .attr("x", textX)
      .attr("y", textY)
      .style("stroke", textStroke)
      .style("stroke-width", 0)
      .style("font-size", fontSize)
      .style("fill", textColor);
  });
</script>
</body>
</html>

```

Listing 9.5 starts with some boilerplate code and initialization of some JavaScript variables. The next portion of code handles the reading and parsing of the data from the CSV file `Voronoi.csv` and then populates the JavaScript vertices array, as shown here:

```

d3.text("Voronoi.csv", function(datasetText) {
  var vertices = d3.csv.parseRows(
    datasetText,
    function(pt) {
      return [parseFloat(pt[0]),parseFloat(pt[1])];
    }
  )

  // D3 code for rendering the data
})

```

The remaining D3 code in Listing 9.5 is very familiar to you and straightforward: create an `<svg>` element and append it to the `<div>` element whose `id` attribute has the value `chart`, iterate through the data in the `vertices` JavaScript array and create corresponding SVG `<path>` elements, add SVG `<circle>` elements on the vertices, and display a text string using the text-related JavaScript variables.

Listing 9.6 displays the contents of the Perl script `CSVVoronoi1.pl` that generates the data in the CSV file `Voronoi1.csv`.

**LISTING 9.6 *CSVVoronoi1.pl***

```
#!/usr/bin/perl
use strict;
use warnings;

my $count = 0;
my $xRange = 600;
my $yRange = 400;
my $maxCount = 100;

for ($count = 0; $count < $maxCount; $count++) {
    my $xRand = int(rand($xRange));
    my $yRand = int(rand($yRange));

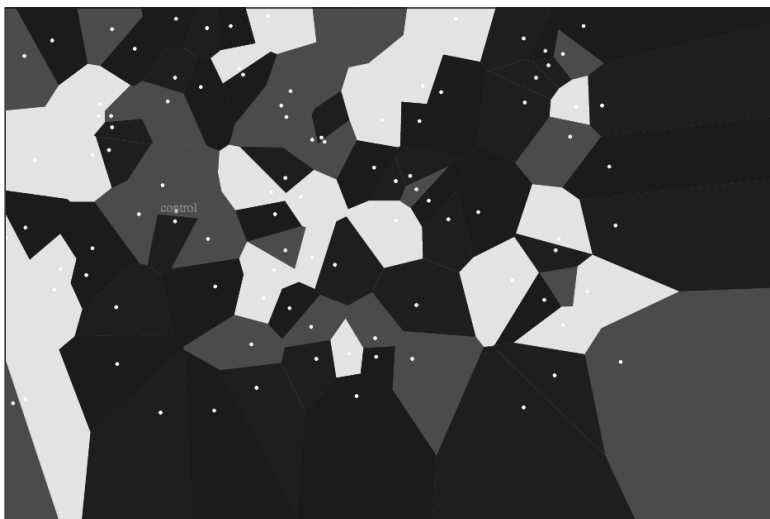
    print $xRand . "," . $yRand . "\n";
}
```

Listing 9.6 starts by defining several variables. Next, a loop generates a pair of random numbers and then prints that pair of numbers as a single row of data. If you have Perl installed on your machine, open a command shell and launch the Perl script in Listing 9.6. Redirect the output to create the contents of the CSV file `Voronoi1.csv` that is referenced in Listing 9.5 as follows:

```
perl CSVVoronoi.pl > Voronoi.csv
```

Figure 9.5 displays a Voronoi diagram in D3 that is defined in Listing 9.5.

The next section in this chapter discusses two toolkits that are extensions of D3 and a D3 plugin (written by Mike Bostock).



**FIGURE 9.5** Rendering a Voronoi Diagram in D3.

## TOOLKITS THAT ARE D3 EXTENSIONS

---

There are several useful toolkits available that are extensions of D3, some of which are listed here:

- ChartBuilder
- CrossFilter.js
- Dc.js
- Rickshaw

The following subsections briefly discuss these D3 extensions and provide links where you can obtain additional information.

### The ChartBuilder Extension

---

ChartBuilder was turned into an open-source project on July 30, 2013, and its homepage is can be found at <https://github.com/Quartz/Chartbuilder/>. Chartbuilder provides the user an export interface, whereas Gneisschart is the charting framework.

An interesting article about ChartBuilder (“How to Turn Everyone in Your Newsroom into a Graphics Editor,”) and some visual samples and use cases is available here at <http://www.niemanlab.org/2013/07/how-to-turn-everyone-in-your-newsroom-into-a-graphics-editor/>.

### The CrossFilter Extension

---

Crossfilter is designed for examining large multivariate datasets with support for extremely fast coordinated views, and its homepage is found at <http://square.github.io/crossfilter/>. Crossfilter was created to support analytics for Square. The CrossFilter APIs include methods for filtering data and map-reduce operations; they are described here:

*<https://github.com/square/crossfilter/wiki/API-Reference>.*

### The dc.js Extension

---

The dc.js toolkit (which depends on `crossfilter.js`) is also useful for exploring large multidimensional datasets, and you can find a code sample and a demo on its homepage at <http://nickqizhu.github.io/dc.js/>. According to the homepage of this project:

The main objective of this project is to provide an easy yet powerful JavaScript library, which can be utilized to perform data visualization and analysis in browsers as well as on mobile devices.

There is also a dc.js user group in case you need to post questions regarding d3.js, found at: <https://groups.google.com/forum/?fromgroups#!forum/dc-js-user-group>. Visit the homepage for links to the wiki page and the API documentation.

## Rickshaw

Rickshaw is a toolkit that provides a layer over D3, and its homepage is found at <http://code.shutterstock.com/rickshaw/>.

Listing 9.7 displays the contents of `Rickshaw1.html` that illustrate how to render an area chart using the Rickshaw toolkit.

### LISTING 9.7 *Rickshaw1.html*

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Rickshaw Area Chart</title>
    <script src="http://d3js.org/d3.v3.min.js"></script>
    <script src="rickshaw.min.js"></script>
  </head>

  <body>
    <div id="chart"></div>

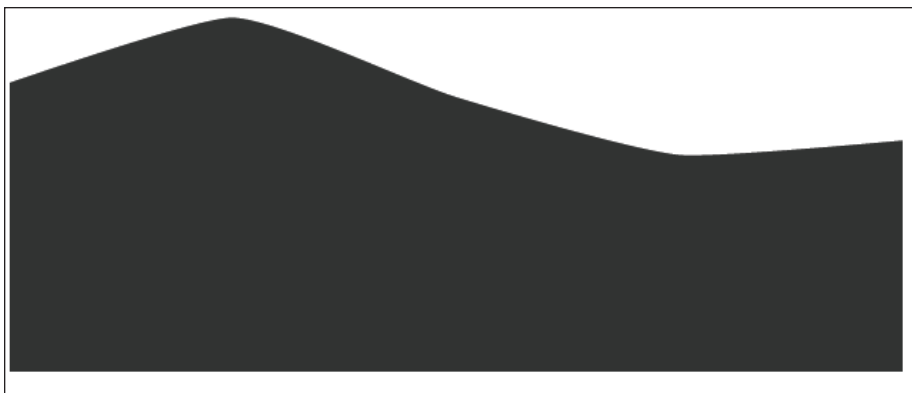
    <script>
      var graph = new Rickshaw.Graph({
        element: document.querySelector("#chart"),
        width: 500,
        height: 200,
        series: [{
          color: 'steelblue',
          data: [
            { x: 0, y: 40 },
            { x: 1, y: 49 },
            { x: 2, y: 38 },
            { x: 3, y: 30 },
            { x: 4, y: 32 } ]
        }]
      });

      graph.render();
    </script>
  </body>
</html>
```

Listing 9.7 starts with boilerplate code followed by a `<div>` element where the chart-related data is appended (represented as SVG elements). The next portion of code defines a JavaScript `graph` variable that is populated with JSON-based data that specifies various attributes, such as the width and height of the area chart. The actual data points for the area chart are specified as JSON-based values in the `data` property of the `graph` variable. Finally, the area chart is rendered with the following code snippet:

```
graph.render();
```

Figure 9.6 displays the area chart that is defined in Listing 9.7.



**FIGURE 9.6** An Area Chart Rendered with the Rickshaw Toolkit.

## D3 AND OTHER TOOLKITS

---

As you saw in the introduction, D3 can be used with many JavaScript toolkits. This chapter provides very limited information about toolkits that are outside the scope of this introductory D3 book. However, Shirley Wu describes some of the “gotchas” that she encountered in the process of combining D3 with BackboneJS (it’s good to know what works as well as what does not work) in this blog post:

*<http://shirley.quora.com/Marrying-Backbone-js-and-D3-js>.*

Miles McCrocklin wrote a five-part series that discusses how to combine D3 with Prototype, AngularJS, EmberJS, and BackboneJS, found at <http://blocks.org/milroc/5522467>. The following link uses BackboneJS and D3 to create bar charts: <http://drsm79.github.com/Backbone-d3/bar.html>. Another link that shows you how to use AngularJS with D3 can be found at <https://github.com/btford/angular-d3-demo>.

Perform an Internet search, and you will find links about various toolkits along with code samples that illustrate how to use them with D3.

## D3 PLUGINS

---

`Cubism.js` is a D3 plugin (also created by Mike Bostock) that creates time-series charts, and its homepage can be found at <https://github.com/square/cubism>. Alternately, you can install Cubism on your machine with this command:

```
npm install -g cubism
```

An interesting stock-based example using `Cubism.js` can be found at <http://bost.ocks.org/mike/cubism/intro/demo-stocks.html>. This demo takes a few

moments to load the data. Move your mouse horizontally and watch how the sliding vertical line shows the changes in each stock at the current time period.

A list of available D3 plugins is: <https://github.com/d3/d3-plugins>. If you are interested in creating your own D3 plugins, an article by Mike Bostock that provides good information regarding custom D3 plugins can be found at <http://bost.ocks.org/mike/chart/>.

## **DVL(DYNAMIC VISUALIZATION LEGO) FOR DATA VISUALIZATION**

---

DVL (Dynamic Visualization LEGO) is a functionally reactive framework that addresses the challenge of creating highly interactive visualizations and UIs (User Interfaces), which means that it can respond quickly to changes in the data that are rendered in a visualization. DVL is specifically designed to assist in creating exploratory data visualizations using real-time data stored in databases.

DVL has the notion of workers, which are functions that transform to variables. Workers are interconnected to create a data dependency graph, and DVL maintains the topological ordering of that dependency graph using an online topological sorting algorithm. The DVL metadata computed in the graph enables DVL to propagate changes through the graph in an efficient manner without violating any of the stated dependencies.

DVL also supports various output modules that facilitate the binding of data to the DOM via D3.js. These modules can be used to create visualizations that automatically react to their inputs and perform updates and animations accordingly. This functionality is especially useful for synchronizing visualizations that bind directly to real-time/streaming data that is continually being updated.

Some of the future direction/functionality includes improved UI and Vis components built specifically to integrate well with DVL. Although many of those components are already developed and used at MetaMarkets, the components will be generalized before they are included in DVL.

DVL has a minimalist design; it's essentially the dependency graph manager as well as the UI component layer that are built on top of D3.js and jQuery. The goal is to expand DVL so that it can be used as a general framework that can integrate with other systems. Currently DVL is used heavily with Druid (another product of MetaMarkets), which involves a small set of proprietary functions to communicate between DVL and Druid.

DVL is an open-source project created at Metamarkets by Vadim Ogievetsky (a contributor to the D3.js toolkit). More information and downloadable code at <https://github.com/vogievetsky/dvl>.

## **VEGA: A VISUALIZATION GRAMMAR**

---

Vega is a toolkit that is based on D3, and according to the Vega Website:

Vega uses a declarative format for creating, saving, and sharing visualization designs without programming. With Vega you can describe

data visualizations in a JSON format and generate reusable chart components that flexibly render with either HTML5 Canvas or SVG.

You can find a download link and additional information about Vega at <http://trifacta.github.io/vega/>.

## NVD3

---

NVD3 is a charting library that is based on D3, and its homepage is <http://nvd3.org/>. The goal of NVD3 is to create a collection of reusable components in D3. If you prefer working with Python, you might be interested in this open-source project that provides a Python wrapper around NVD3: <https://github.com/areski/python-nvd3>.

## DEXCHART: REUSABLE CHARTS

---

DexChart is another charting library with the goal of providing reusable charts and components, as well as a framework for interconnecting these charts via listeners.

You can DexCharts (along with code samples) here: <https://github.com/PatMartin/DexCharts>.

The remaining portion of this chapter discusses other D3 APIs that involve more advanced functionality.

## R PROGRAMMING WITH D3-BASED TOOLKITS

---

The following description of R is from its Wikipedia page:

R is a free software programming language and software environment for statistical computing and graphics. The R language is widely used among statisticians and data miners for developing statistical software and data analysis. Polls and surveys of data miners are showing R's popularity has increased substantially in recent years.

You can read the full entry in Wikipedia at [http://en.wikipedia.org/wiki/R\\_\(programming\\_language\)](http://en.wikipedia.org/wiki/R_(programming_language)).

There are toolkits available that provide an R interface with various D3-related toolkits. For example, rCharts is an R interface for NVD3, Polycharts, MorrisJs (with future support for Rickshaw, DexCharts, and dc.js), and its homepage is found at <http://ramnathv.github.io/rCharts/r2js>. There is also open source-toolkit rNVD3 that provides an R interface for NVD3, and its homepage is <http://ramnathv.github.io/rNVD3/>.

## ADDITIONAL D3 APIS

---

D3 supports additional APIs that are beyond the scope of this book, but it's good to be aware of their existence in case you need their functionality in the



future. This section briefly lists some of those APIs (but without examples). One useful D3 API is called Cross Filters, which provides multidimensional filtering of data on coordinated views. An interesting project and illustrative demo are available on github at <http://square.github.io/crossfilter/>.

D3 provides projections via `d3.geo.projection`, many of which involve projecting map-related data onto a two-dimensional plane. Details regarding the D3 geo-related API and various examples are found at <https://github.com/mbostock/d3/wiki/Geo-Projections>.

## THE D3 BRUSHES API

---

D3 Brushes enable you to highlight a region of a chart or graph in data visualization. Although this functionality is beyond an introductory-level topic, it's worth looking at some examples so you can get an idea of what you can do with D3 Brushes.

The following link provides a good example of using a D3 brush component with zoom functionality to provide contextual information: <http://bl.ocks.org/mbostock/1667367>.

You can use D3 brushes with so-called quad trees. Perform a mouse down followed by a mouse drag to highlight a rectangular region in this quad tree:

*<http://bl.ocks.org/mbostock/4343214>.*

## D3 AND HTML5 WEB AUDIO

---

The HTML5 Web Audio APIs are available in Chrome and Firefox browsers, and they make it possible to render audio waves in HTML5 Canvas to create a chart-like effect in real time. The level of support for HTML5 Web Audio APIs in different browsers is summarized here: <http://caniuse.com/#feat=audio-api>.

If audio-related functionality is of interest to you, then you will be pleased to discover that the same functionality is possible with D3, and an example is here:

*<http://d3-spectrum.herokuapp.com/>.*

Perform an Internet search to find other interesting code samples that use D3 to render audio files in browsers.

## WHAT ABOUT D3 FOR THREE-DIMENSIONAL GRAPHICS AND ANIMATION?

---

As you now know, SVG is an XML-based technology that supports 2D shapes and animation effects. You have seen code samples that use JavaScript to programmatically create and render SVG-based 2D shapes to create 3D effects. However, SVG does not have any notion of whether or not a 2D shape is rendered in 3D space to create a 3D effect. This limitation to 2D is also true of D3 because of its direct dependency on SVG. Although D3 supports HTML5 Canvas, this support is also limited to 2D shapes.

However, there are several 3D technologies that run mainly in desktop browsers, and in some cases on mobile devices, including:

- X3D
- WebGL
- Three.js (a JavaScript toolkit on top of WebGL)
- tQuery (jQuery and Three.js)
- CSG

X3D is the ISO standard XML-based file format for representing 3D computer graphics, the successor to VRML (Virtual Reality Modelling Language). The X3D homepage is <http://www.x3dom.org/>. A simple example that illustrates how to combine X3D with D3 is found at <http://bl.ocks.org/ZJONSSON/1291672>.

WebGL supports 2D/3D graphics and animation effects. However, WebGL support is primarily for laptops and desktops with limited support for mobile devices. Both Chrome Beta and Firefox on mobile devices support some features of WebGL, and you can check the documentation for these mobile browsers to get more detailed information.

Keep in mind that WebGL is probably an order of magnitude more difficult than D3. One of the appendices for this book provides a short introduction to WebGL, and you can perform an Internet search if you want to read other articles with additional information about WebGL.

ThreeJS is a JavaScript toolkit that provides a layer of abstraction over WebGL, and its homepage is <https://github.com/mrdoob/three.js/>. An example of rendering a 3D map with D3 and Three.js is found at <http://css.dzone.com/articles/render-geographic-information>. Additional examples of using Three.js are here:

*<http://srchea.com/blog/2012/02/terrain-generation-the-diamond-square-algorithm-and-three-js/>, <http://www.smartjava.org/content/threejs-render-real-world-terrain-heightmap-using-open-data>, and <http://grahamweldon.com/posts/view/3d-terrain-generation-with-three-js>.*

## OTHER D3 RESOURCES

There are various Websites devoted to D3, and this section provides you with a few of them. The best thing for you to do is to navigate to these Websites to see which ones are most closely aligned with the aspects of D3 that interest you.

A very extensive D3.js gallery (most examples are by Mike Bostock) is found at <http://biovisualize.github.com/d3visualization/>.

A Google Docs spreadsheet with details about the samples in the D3.js gallery is:

*<https://docs.google.com/spreadsheet/ccc?key=0AqMEGBUNwXeHdHpQNIVuY29SUE5BSXVtS3JueGlNYVE#gid=0>.*

Christophe Viau's D3 slides are found here: [http://biovisualize.github.com/start\\_with\\_d3/](http://biovisualize.github.com/start_with_d3/).

There are various ways to contribute to D3 (the following list is from Christophe Viau):

- Gist
- bl.ocks
- jsFiddle
- Plunker

You can perform a Google search to obtain more information about the items in the preceding list.

## ADDITIONAL CODE SAMPLES ON THE CD



The HTML Web page `USMapAnnualRainfall1Mouse1.html` adds extra color and mouse-related functionality: whenever a `mouseover` event over a state occurs, the color of the state changes from a red-based gradient to a blue-based gradient and also scales down the size of the state (similar to the other map code sample). This simple enhancement in the functionality of the original code sample is admittedly small, yet it is the type of functionality with a nuance that can provide a better user experience.

If you like Voronoi diagrams, then you might also like the two HTML Web pages `CSVVoronoi1BlurFilter1.html` and `CSVVoronoi1Pattern1.html` that add an SVG `<filter>` element based on Gaussian blur filter and an SVG `<pattern>` element, respectively, to the HTML Web page `CSVVoronoi1.html`.

The pair of HTML Web pages `Force1BlurFilter1.html` and `Force1BlurPattern1.html` both enhance the HTML Web page `Force1.html` by adding a Gaussian blur filter and an SVG `<pattern>` element, respectively. Similarly, the HTML Web pages `QBezier1Plot1BlurFilter1.html` and `SimpleTree1BlurFilter1.html` contain Gaussian blur filters, and they are the counterparts to the HTML Web pages `QBezier1Plot1BlurFilter1.html` and `SimpleTree1BlurFilter1.html`.

## SUMMARY

This chapter showed you how to use an assortment of D3 APIs for rendering United States maps and tree-like data structures. You also learned about force diagrams and Voronoi diagrams. Finally, you learned about the D3 plugin Cubism (written by Mike Bostock) as well as Rickshaw and `dc.js`, which are two D3 extensions.

# *HTML5 MOBILE APPLICATIONS ON ANDROID AND IOS*

**T**his chapter shows you how to create HTML5-based hybrid mobile applications for Android and iOS. The code samples in this chapter contain HTML5 and various combinations of HTML5, CSS3, SVG, and D3.

The first part of this chapter provides an overview of how to develop hybrid Android applications. The code samples in this section use the same code that you have seen in earlier chapters, and they show you how to create the hybrid Android mobile applications that will enable you to create the same screenshots. If you feel ambitious, you can create Android-based mobile applications for all the code samples in this book!

The second part of this chapter contains Android-based code samples that show you how to combine native Android applications with CSS3, SVG, and HTML5 Canvas. This section contains an example of rendering a mouse-enabled multiline graph whose values can be updated whenever users click on the button that is rendered underneath the line graph. Keep in mind that the discussion following the code samples moves quickly because the HTML Web pages contain simple markup, the CSS3 selectors contain code that you have seen in earlier chapters, and the SVG shapes are discussed in Chapter 4.

The third part of this chapter provides a quick overview of Apache Cordova, formerly known as PhoneGap, which is a popular cross-platform toolkit for developing mobile applications. In 2011 Adobe acquired Nitobi, the company that created PhoneGap, and shortly thereafter Adobe open sourced PhoneGap. This section explains how to use PhoneGap to create hybrid mobile applications. You will learn how to create a PhoneGap-based Android application that renders CSS3-based animation effects, and you can deploy this mobile application to Android-based mobile devices. Keep in mind that the term PhoneGap is used in this chapter (and sometimes even by Adobe) to refer to

Apache Cordova because the name change is still quite recent, and there is a huge installed code base that still refers to Apache Cordova as PhoneGap.

The final part of this chapter shows you some of the functionality of Xcode 5, and you will see the steps for creating a Hello World mobile application for iOS devices. If you are unfamiliar with any of the mobile platforms in this chapter you can still work through the examples because they consist of HTML5-based code, and the sequence of steps for creating HTML5-based mobile applications on a mobile platform is essentially independent of the actual code.

## HTML5/CSS3 AND ANDROID APPLICATIONS

If you are unfamiliar with Android, you can read the appendix for this book that contains a concise overview of the Android-specific concepts in the code samples in this chapter. You can refer to the appropriate section whenever you encounter an Android concept that is not clear to you.

The code sample in this section shows you how to launch an HTML5 Web page (which also references a CSS3 stylesheet) inside an Android application. The key idea consists of three steps:

1. modify the Android Activity class to instantiate an Android Web-View class, along with some JavaScript-related settings
2. reference an HTML5 Web page that is in the `assets/www` subdirectory of the Android project
3. copy the HTML5 Web page, CSS stylesheets, and JavaScript files into the `assets/www` subdirectory of the Android project

In Step 3, you will probably create a hierarchical set of directories that contain files that are of the same type (HTML, CSS, or JavaScript), in much the same way that you organize your files in a Web application.

Now launch Eclipse and create an Android project called `AndroidCSS3`, making sure that you select Android version 3.1 or higher, which is necessary to render CSS3-based effects.

After you have created the project, let's take a look at three files that contain the custom code for this Android mobile application. Listing 10.1, 10.2, and 10.3 display the contents of the project files `activity_main.xml`, `AndroidCSS3.html`, and `AndroidCSS3Activity.java`.

### LISTING 10.1 `activity_main.xml`

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <WebView android:id="@+id/webview"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent">
```

```

</WebView>
</LinearLayout>

```

Listing 10.1 specifies a `LinearLayout` that contains an Android `WebView` that will occupy the entire screen of the mobile device. This is the behavior that we want to see because Android default browser is rendered inside the Android `WebView`.

#### **LISTING 10.2** *AndroidCSS3.html*

```

<!doctype html>
<head>
  <meta charset="utf-8" />
  <title>CSS Radial Gradient Example</title>
  <link href="AndroidCSS3.css" rel="stylesheet">
</head>

<body>
  <div id="outer">
    <div id="radial1">Text1</div>
    <div id="radial2">Text2</div>
    <div id="radial3">Text3</div>
    <div id="radial4">Text4</div>
  </div>
</body>
</html>

```



Listing 10.2 is a straightforward HTML Web page that references a CSS stylesheet `AndroidCSS3.css` (that is available on the CD) with an HTML `<div>` element (whose `id` attribute has value `outer`) that serves as a container for four more HTML `<div>` elements.

The CSS stylesheet `AndroidCSS3.css` contains a CSS selector for styling the HTML `<div>` element whose `id` has value `outer` followed by four CSS selectors, `#radial1`, `#radial2`, `#radial3`, and `#radial4`, that are used to style the corresponding HTML `<div>` elements in Listing 10.2. The contents of these selectors ought to be very familiar (you can review the material for CSS3 gradients in an earlier chapter), so we will not cover their contents in this section.

#### **LISTING 10.3** *AndroidCSS3Activity.java*

```

package com.iquarkt.css3;

import android.app.Activity;
import android.os.Bundle;
import android.view.KeyEvent;
import android.view.Menu;
import android.webkit.WebView;
import android.webkit.WebViewClient;

public class AndroidCSS3Activity extends Activity
{
    WebView mWebView;

```

```

/** Called when the activity is first created. */
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    mWebView = (WebView) findViewById(R.id.webview);
    mWebView.setWebViewClient(new ChangeURLClient());
    mWebView.getSettings().setJavaScriptEnabled(true);
    mWebView.getSettings().setDomStorageEnabled(true);
    mWebView.loadUrl(
        "file:///android_asset/www/FollowTheMouse1Anim1.html");
}

@Override
public boolean onCreateOptionsMenu(Menu menu) {
    // Inflate the menu; this adds items to the
    // action bar if it is present.
    getMenuInflater().inflate(R.menu.main, menu);
    return true;
}

public boolean onKeyDown(int keyCode, KeyEvent event) {
    if ((keyCode==KeyEvent.KEYCODE_BACK)&&mWebView.canGoBack()) {
        mWebView.goBack();
        return true;
    }

    return super.onKeyDown(keyCode, event);
}

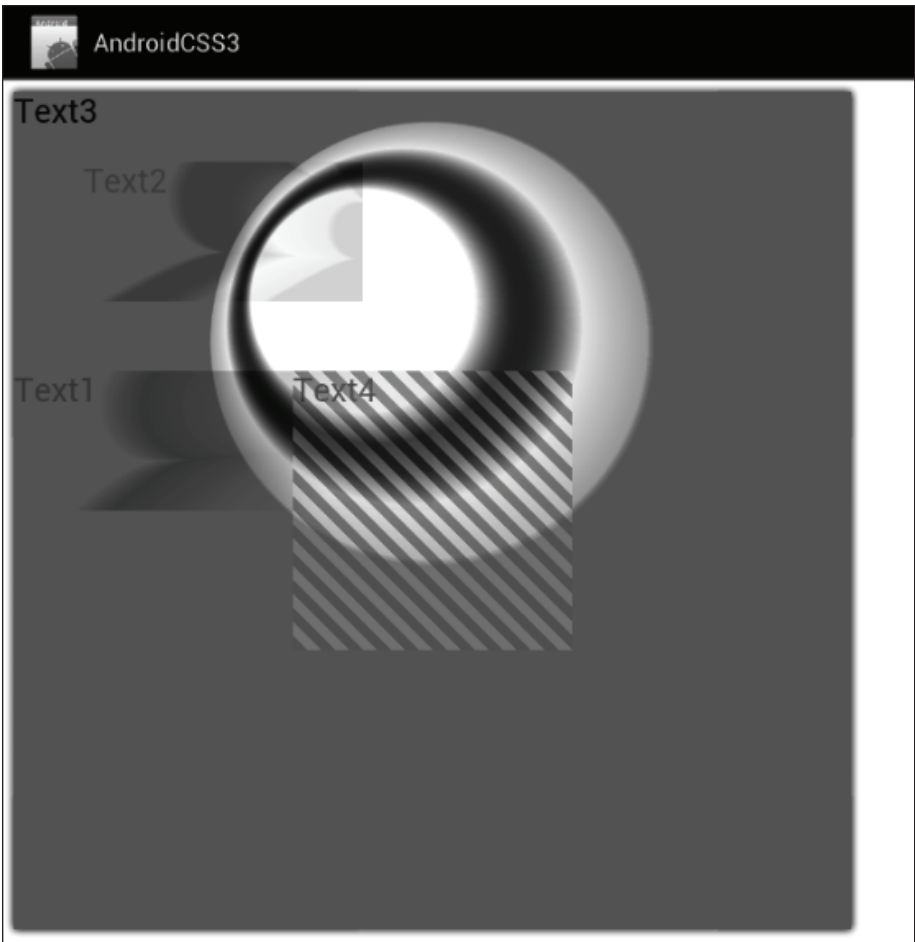
private class ChangeURLClient extends WebViewClient {
    public boolean shouldOverrideUrlLoading(WebView view, String url)
    {
        System.out.println("URL: " + url);
        view.loadUrl("javascript:changeLocation('"+url+"'");
        return true;
    }
}
}

```

Listing 10.3 defines a Java class `AndroidCSS3Activity` that extends the standard `Android Activity` class. This class contains the `onCreate()` method that points to the XML document `activity_main.xml` (displayed in Listing 10.2) so we can get a reference to its `WebView` child element via `R.id.webview` (which is the reference to the `WebView` element in Listing 10.2), as shown here:

```
WebView webview = (WebView) this.findViewById(R.id.webview);
```

Next, the `webSettings` instance of the `WebSettings` class enables us to set various properties, as shown in the commented lines of code in Listing 10.4.



**FIGURE 10.1** A CSS3-based 3D Cube on an Asus Prime Tablet with Android ICS.

The final line of code loads the contents of the HTML Web page `AndroidCSS3.html` (which is in the `assets/www` subdirectory), as shown here:

```
webView.loadUrl("file:///android_asset/AndroidCSS3.html");
```

Figure 10.1 displays a CSS3-based Android application on an Asus Prime tablet with Android ICS.

## SVG AND ANDROID APPLICATIONS

The example in this section shows you how to create an Android mobile application that renders SVG code that is embedded in an HTML5 Web page. Now launch Eclipse and create an Android project called `AndroidSVG1`;



make sure that you select Android version 3.1 or higher, which is necessary to render SVG elements.

The example in the preceding section contains four custom files, whereas the Android/SVG example in this section contains two files with custom code: the HTML5 Web page `AndroidSVG1.html` in Listing 10.4 and the Java class `AndroidSVG1.java`, which are available on the CD.



#### **LISTING 10.4** *AndroidSVG1.html*

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8" />
    <title>Android and SVG</title>
  </head>

  <body>
    <h1>HTML5/SVG Example</h1>
    <svg>
      <ellipse cx="300" cy="50" rx="80" ry="40"
        fill="#ff0" stroke-dasharray="8 4 8 1"
        style="stroke:red;stroke-width:4;"/>

      <line x1="100" y1="20" x2="300" y2="350"
        stroke-dasharray="8 4 8 1"
        style="stroke:red;stroke-width:8;"/>

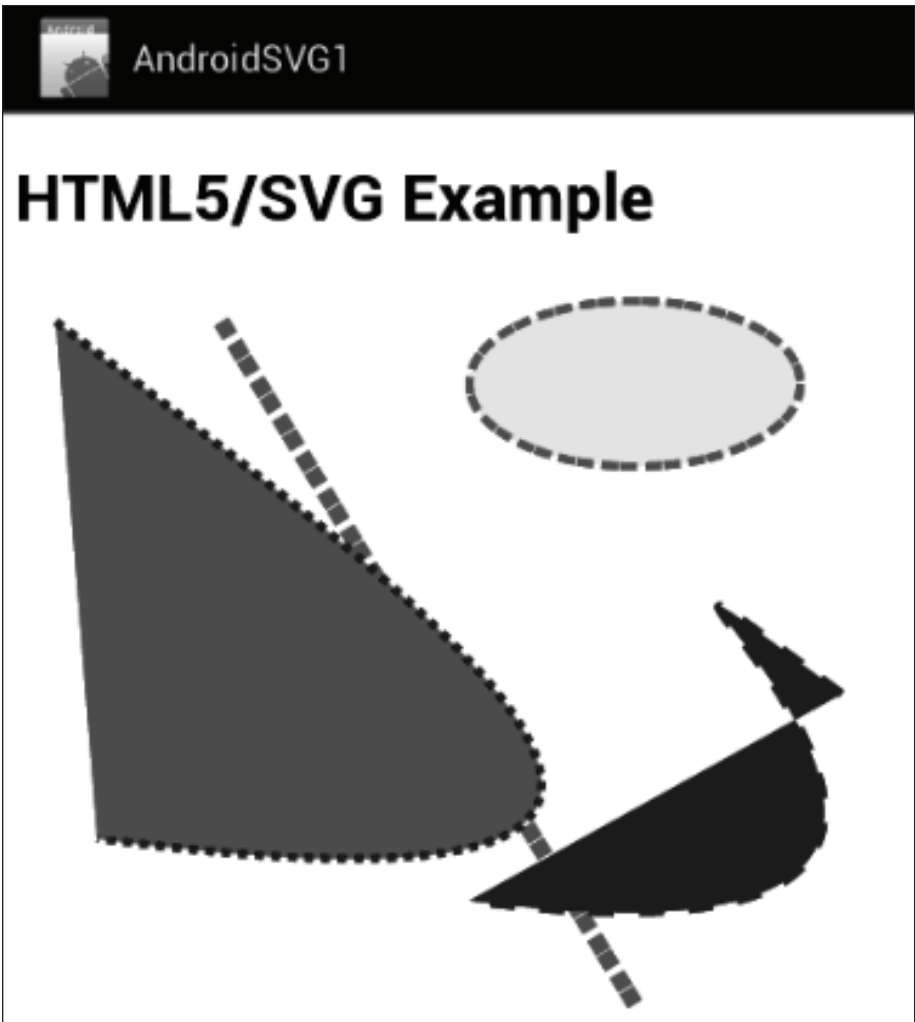
      <g transform="translate(20,20)">
        <path
          d="M0,0 C200,150 400,300 20,250"
          fill="#f00"
          stroke-dasharray="4 4 4 4"
          style="stroke:blue;stroke-width:4;"/>
        </g>

      <g transform="translate(200,50)">
        <path
          d="M200,150 C0,0 400,300 20,250"
          fill="#00f"
          stroke-dasharray="12 12 12 12"
          style="stroke:blue;stroke-width:4;"/>
        </g>
      </svg>
    </body>
  </html>
```

Listing 10.4 is an HTML Web page that contains an SVG document with the definitions for an ellipse, a line segment, and two cubic Bezier curves. Appendix A contains examples of these 2D shapes (among others), and you can review the appropriate material if you need to refresh your memory.

The Java class `AndroidSVG1Activity.java` is omitted, but its contents are very similar to Listing 10.3, and the complete source code is available on the CD.





**FIGURE 10.2** An SVG-based Android Application on an Asus Tablet with Android ICS.

Figure 10.2 displays an SVG-based Android application on an Asus Prime tablet with Android ICS.

## HTML5 CANVAS AND ANDROID APPLICATIONS

In addition to rendering CSS3-based effects and SVG documents, you can also render Canvas-based 2D shapes in an Android application. Launch Eclipse and create an Android project called `AndroidCanvas1`; make sure that you select Android version 3.1 or higher, which is necessary to render SVG elements.

The example in this section contains one custom file called `AndroidCanvas1.html`, which is displayed in Listing 10.5.

**LISTING 10.5 *AndroidCanvas1.html***

```

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="utf-8" />
  <title>Android and HTML5 Canvas</title>

  <link href="AndroidCSS3.css" rel="stylesheet"
        type="text/css">
</head>

<script>
function draw() {
  var basePointX   = 10;
  var basePointY   = 80;
  var currentX     = 0;
  var currentY     = 0;
  var startAngle   = 0;
  var endAngle     = 0;
  var radius       = 120;
  var lineLength   = 200;
  var lineWidth    = 1;
  var lineCount    = 200;
  var lineColor    = "";

  var hexArray     = new Array(
    '0','1','2','3','4','5','6','7',
    '8','9','a','b','c','d','e','f');

  var can = document.getElementById('canvas1');
  var ctx = can.getContext('2d');

  // render a text string...
  ctx.font = "bold 26px helvetica, arial, sans-serif";
  ctx.shadowColor = "#333333";
  ctx.shadowOffsetX = 2;
  ctx.shadowOffsetY = 2;
  ctx.shadowBlur = 2;
  ctx.fillStyle = 'red';
  ctx.fillText("HTML5 Canvas/Android", 0, 30);

  for(var r=0; r<lineCount; r++) {
    currentX = basePointX+r;
    currentY = basePointY+r;
    startAngle = (360-r/2)*Math.PI/180;
    endAngle   = (360+r/2)*Math.PI/180;

    // render the first line segment...
    lineColor = '#' + hexArray[r%16] + '00';
    ctx.strokeStyle = lineColor;
    ctx.lineWidth   = lineWidth;

    ctx.beginPath();
    ctx.moveTo(currentX, currentY+2*r);
    ctx.lineTo(currentX+lineLength, currentY+2*r);
    ctx.closePath();
  }
}

```

```

        ctx.stroke();
        ctx.fill();

        // render the second line segment...
        lineColor = '#' + '0' + hexArray[r%16] + '0';
        ctx.beginPath();
        ctx.moveTo(currentX, currentY);
        ctx.lineTo(currentX+lineLength, currentY);
        ctx.closePath();
        ctx.stroke();
        ctx.fill();

        // render the arc...
        lineColor = '#' + '00' + hexArray[(2*r)%16];
        ctx.beginPath();
        ctx.fillStyle = lineColor;
        ctx.moveTo(currentX, currentY);
        ctx.arc(currentX, currentY, radius,
                startAngle, endAngle, false);
        ctx.closePath();
        ctx.stroke();
        ctx.fill();
    }
}
</script>

<body onload="draw()">
  <canvas id="canvas1" width="300px" height="200px"></canvas>
</body>
</html>

```

Listing 10.5 contains some boilerplate HTML markup and a JavaScript function `draw()` that is executed when the Web page is loaded into the Android browser. The `draw()` function contains JavaScript code that draws a set of line segments and arcs into the HTML5 `<canvas>` element whose `id` attribute has value `canvas1`. You can review the HTML5 Canvas code samples in the appendix that have similar functionality if you don't remember the details of the syntax of this JavaScript code.

Figure 10.3 displays a Canvas-based Android application on an Asus Prime tablet with Android ICS.

## ANDROID AND HTML5 CANVAS MULTILINE GRAPHS

Although Android does not have built-in support for rendering charts and graphs, you can create them using Canvas-based code that is very similar to the code in the previous section.

Launch Eclipse and create an Android project called `AndroidCanvasMultiLine2`; make sure that you select Android version 3.1 or higher. Listing 10.6 displays portions of the HTML5 Web page `AndroidCanvasMultiLine2.html` that contain JavaScript code for rendering multiple line graphs using HTML5 Canvas.



**FIGURE 10.3** A Canvas-based Android Application on an Asus Tablet with Android ICS.

#### **LISTING 10.6** *AndroidCanvasMultiLine2.html*

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8" />
    <title>HTML5 Canvas Line Graphs</title>

    <script>
      // JavaScript variables omitted for brevity
      var lineCount      = 3;
      var barHeights     = new Array(barCount);
      var barHeights2    = new Array(barCount);
      var barHeights3    = new Array(barCount);
      var currHeights    = new Array(barCount);
      var multiLines     = new Array(lineCount);
      var fillColors     = new Array(
        "#F00", "#FF0", "#0F0", "#00F");
      var elem, context, gradient1;

      function drawGraph() {
        // make sure that you clear the canvas
        // before drawing a new set of rectangles
        context.clearRect(0, 0, elem.width, elem.height);

        randomBarValues();
```

```

        drawAndLabelAxes();
        drawElements();
    }

    function randomBarValues() {
        for(var i=0; i<barCount; i++) {
            barHeight = maxHeight*Math.random();
            barHeights[i] = barHeight;

            barHeight = maxHeight*Math.random();
            barHeights2[i] = barHeight;

            barHeight = maxHeight*Math.random();
            barHeights3[i] = barHeight;
        }

        multiLines[0] = barHeights;
        multiLines[1] = barHeights2;
        multiLines[2] = barHeights3;
    }

    function drawElements() {
        for(var h=0; h<multiLines.length; h++) {
            currHeights = multiLines[h];

            currentX = leftBorder;
            //currentY = maxHeight-barHeights[0];
            currentY = maxHeight-currHeights[0];

            // draw line segments...
            for(var i=0; i<barCount; i++) {
                context.beginPath();
                context.moveTo(currentX, currentY);
                currentX = leftBorder+i*barWidth;
                //currentY = maxHeight-barHeights[i];
                currentY = maxHeight-currHeights[i];

                context.shadowColor    = "rgba(100,100,100,.5)";
                context.shadowOffsetX = 3;
                context.shadowOffsetY = 3;
                context.shadowBlur    = 5;

                context.lineWidth      = 4;
                context.strokeStyle = fillColors[i%4];
                context.lineCap        = "miter"; // "round";

                context.lineTo(currentX, currentY);
                context.stroke();
            }
        }

        drawBarText();
    }
}
</script>
</head>

```

```

<body onload="drawGraph();" >
  <header>
    <h1>HTML5 Canvas Line Graphs</h1>
  </header>

  <div>
    <canvas id="myCanvas" width="500" height="300">No support for
Canvas
  </canvas>
</div>

  <input type="button" onclick="drawGraph();return false"
    value="Update Graph Values" />
</body>
</html>

```

The JavaScript function `drawGraph()` in Listing 10.6 is invoked when the HTML5 Web page is loaded, and it invokes JavaScript functions to calculate the vertices for three separate line graphs, render labels for the graphs, and then render the three line graphs.

The last step involves a nested loop where the outer loop iterates through the elements of the array `multiLines`, and the inner loop renders one line graph for every iteration through the outer loop.

Figure 10.4 displays a Canvas-based multiline graph Android application on a Nexus S 4G with Android ICS.

The next portion of this chapter delves into PhoneGap, which is a toolkit that automatically creates the lower-level scaffolding that you performed manually earlier in this chapter.

## WHAT IS PHONEGAP?

PhoneGap is an open-source device-agnostic mobile-application development tool that enables you to create cross-platform mobile applications using CSS, HTML, and JavaScript. The PhoneGap homepage provides documentation, code samples, and a download link for the PhoneGap distribution: <http://phonegap.com>.

PhoneGap allows you to create mobile applications using HTML, CSS, and JavaScript, and you can deploy those mobile applications to numerous platforms, including Android, iOS, BlackBerry, and Windows Mobile. You can also create mobile applications that combine PhoneGap with Sencha Touch (another popular framework), but due to space limitations, Sencha Touch is not discussed in this chapter. PhoneGap provides support for touch events, event listeners, rendering images, database access, different file formats (XML and JSON), and Web Services.

---

**NOTE** *If you want to develop iPhone applications, you must have a Macbook or some other OS X machine, all of which are discussed later in this chapter.*

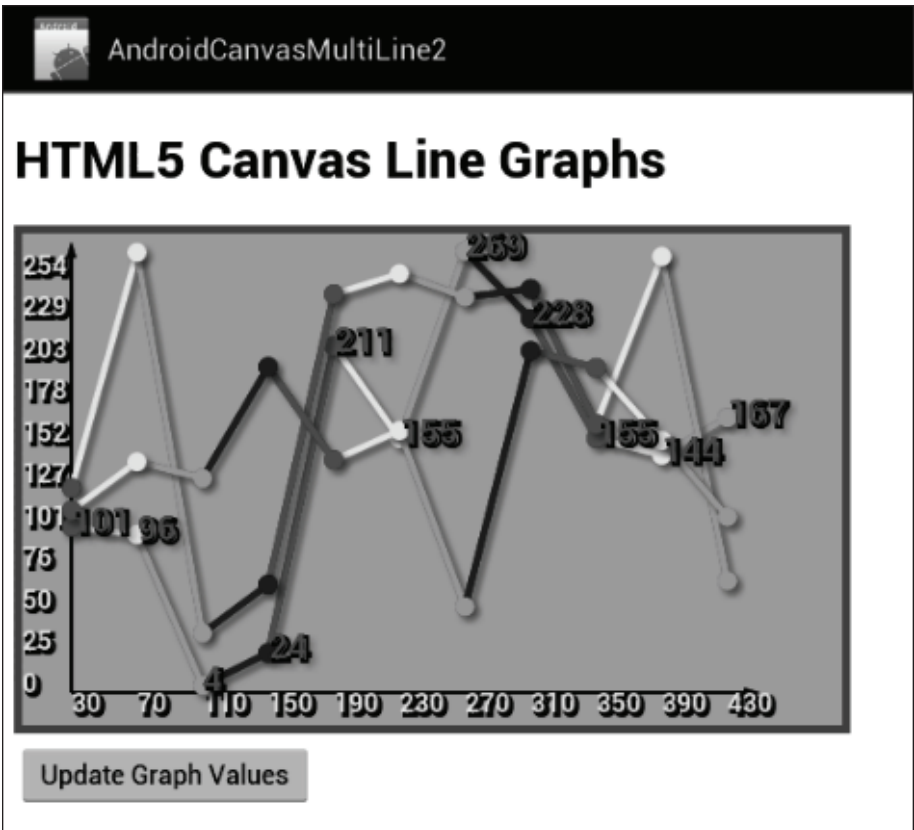


FIGURE 10.4 A Canvas-based Multiline Graph on an Android Smart Phone.

## HOW DOES PHONEGAP WORK?

PhoneGap mobile applications involve a Web view that is embedded in a native shell, and your custom code runs in the Web view. In addition, PhoneGap provides a JavaScript API for accessing native features of a mobile device, and your code can use PhoneGap to access those native features. For example, PhoneGap contains JavaScript APIs for accessing Accelerometer, Camera, Compass, Contacts, Device Information, Events, Geolocation, Media, Notification, and Storage.

Keep in mind that PhoneGap does not provide HTML UI (User Interface) elements, so if you need this functionality in your mobile applications, you can add other toolkits and frameworks such as jQuery Mobile, Sencha Touch, or Appcelerator.

## Software Dependencies for PhoneGap 3.0

To install PhoneGap 3.0, you must first install NodeJS on your machine. NodeJS is required if you want to install any plugins (including the core



plugins) into your application. The development team chose NodeJS to build tools that work in a cross-platform fashion using a single code base.

The examples in this chapter show you how to create hybrid applications using PhoneGap 3.0, which has simplified the process for creating mobile applications compared to earlier versions of PhoneGap. However, if you do not want to install NodeJS, you can download PhoneGap 2.9 (as well as the earlier releases in the 2.x series) that you can install in Eclipse or Xcode. Note that you also have the option of installing PhoneGap 2.9 via NodeJS, whereas NodeJS is a requirement for PhoneGap 3.0.

---

**NOTE** *This chapter does not provide many details about PhoneGap version 2.9 because version 3.0 is the recommended version.*

## CREATING ANDROID HYBRID APPLICATIONS WITH PHONEGAP 3.0

The first step is to install NodeJS on your machine, and you can download a prebuilt installer for your platform at <http://nodejs.org/download/>. After you have successfully installed NodeJS, install PhoneGap with the following command (sudo is not required for Windows machines):

```
sudo npm install -g phonegap
```

You can check the version of PhoneGap with the following command:

```
phonegap -v
```

You will see something like the following (the version number might be different when you perform the installation):

```
3.0.0-0.14.3
```

Now create an Android mobile application via PhoneGap 3.0 by executing the following three commands:

```
phonegap create my-app
cd my-app
phonegap run android
```

After you invoke the third command, you will see something like this on the command line:

```
[phonegap] detecting Android SDK environment...
[phonegap] using the local environment
[phonegap] adding the Android platform...
[warning] missing library cordova/android/3.0.0
[phonegap] downloading https://git-wip-us.apache.org/repos/asf?p=
cordova-android.git;a=snapshot;h=3.0.0;sf=tgz...
[phonegap] compiling Android...
[phonegap] successfully compiled Android app
```

```
[phonegap] trying to install app onto device
[phonegap] no device was found
[phonegap] trying to install app onto emulator
```

If you have an Android mobile device attached to your machine, you will not see the last pair of lines in the preceding output; instead, you will see the HTML5 Web page `index.html` rendered on the screen of the Android mobile device that you have attached to your machine via the USB port.

If size is a concern, keep in mind that PhoneGap generates 300 files in the `my-app` directory, and this generic Hello World application consists of slightly more than seven megabytes. Some of the main files of interest in this mobile application are here:

```
./platforms/android/AndroidManifest.xml
./platforms/android/assets/www/index.html
./platforms/android/assets/www/phonegap.js
./platforms/android/bin/HelloWorld-debug.apk
./platforms/android/src/com/phonegap/hello_world/HelloWorld.java
```

The HTML Web page `index.html` (listed in bold in the preceding list) is the Web page that is launched inside a `WebView` component in the generated Android apk when it is launched in a Simulator or an Android device.

#### NOTE



*PhoneGap applications always have the same filename `index.html`, so to provide multiple PhoneGap project files in the same directory on the CD, the HTML Web page `index.html` for each PhoneGap project is saved in a Web page whose name is the same as the project.*

Listing 10.7 displays the contents of `HelloWorld.java`.

#### LISTING 10.7 *HelloWorld.java*

```
/*
Licensed to the Apache Software Foundation (ASF) under one
or more contributor license agreements. See the NOTICE file
distributed with this work for additional information
regarding copyright ownership. The ASF licenses this file
to you under the Apache License, Version 2.0 (the
"License"); you may not use this file except in compliance
with the License. You may obtain a copy of the License at

    http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing,
software distributed under the License is distributed on an
"AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY
KIND, either express or implied. See the License for the
specific language governing permissions and limitations
under the License.

*/

package com.phonegap.hello_world;
```

```
import android.os.Bundle;
import org.apache.cordova.*;

public class HelloWorld extends DroidGap
{
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);

        // Set by <content src="index.html" /> in config.xml
        super.loadUrl(Config.getStartUrl());
        //super.loadUrl("file:///android_asset/www/index.html")
    }
}
```

Listing 10.7 contains import statements that reference a standard Android class and the classes in the package `org.apache.cordova`. The next portion of code defines the Android class `HelloWorld` that extends the `DroidGap` class. Next, the method `onCreate()` invokes the `onCreate()` method of the super class and then references the HTML Web page `index.html` that is in the following subdirectory of `my-app` (the top-level directory for this project):

```
./platforms/android/assets/www/index.html
```

The following variation of the preceding steps shows you can also specify the package name when you create a PhoneGap Android hybrid mobile application for Android from the command line:

```
cordova create HelloWorld com.example.hello "HelloWorld"
cd HelloWorld
cordova platform add android
cordova build
cordova run android
```

---

**NOTE** *Do not include spaces when you specify “HelloWorld” in the preceding commands because you will get an error with the current release of PhoneGap 3.0.*

If you want to launch the preceding application in the Android simulator, use the following command:

```
cordova emulate android
```

Note that you can specify multiple platforms in a single command when you create a PhoneGap-based mobile application, as shown here:

```
cordova platform add android ios
```

## CREATING IOS HYBRID APPLICATIONS WITH PHONEGAP 3.0

---

Follow the steps in the preceding section for installing NodeJS and PhoneGap 3.0 on your machine. Next, create a new PhoneGap-based project called `my-app2` using the three steps listed in the preceding section, but instead of Android, add an iOS application with the following command:

```
phonegap create my-app2
cd my-app2
phonegap run ios
```

If you see the error message “[Error: Xcode version installed is too old. Minimum: >=4.5.x, yours: Error:],” you need to invoke the following command (adjust the subdirectory of `/Applications` to match the one on your machine):

```
sudo xcode-select --switch /Applications/Xcode5-DP2.app/Contents/Developer
```

The next section describes the prerequisites for deploying mobile applications on an iOS device and contains a code sample.

## REQUIREMENTS FOR DEPLOYING MOBILE APPS TO IOS DEVICES

---

This section shows you how to create iOS mobile applications using PhoneGap; this is exactly the process that was used to create the iOS mobile applications in this book whose screenshots on an iPad3 are included in various chapters. Every iOS mobile application in this book was developed on a Macbook OS X 10.8.2 with Apple’s Xcode 5 and PhoneGap.

Earlier you saw how to create and deploy PhoneGap-based mobile applications to Android, but the situation is more complicated for the iOS platform (with or without PhoneGap).

- Step #1: You need an Apple device (such as a Macbook, Mac Mini, or Mac Pro) with Apple’s Xcode installed to create mobile applications for iOS mobile devices. If you register as a developer, you can download for free, or you can purchase it for \$4.99 in the Apple iStore.
- Step #2: Install PhoneGap 3.0 as described earlier in this chapter.
- Step #3: Register as an Apple Developer (which costs \$99 per year if you want to deploy your iOS mobile applications to iOS devices). However, if you only plan to use the iOS Simulator, you can do so at no charge.

After you have completed the preceding steps, you will be ready to create an iOS mobile application with PhoneGap, which is the topic of the next section.

## RENDERING A CSS3 CUBE ON IOS USING PHONEGAP

Create an Xcode application called ThreeDCube1 by selecting the PhoneGap plugin (make sure that your file names start with an alphabetic character, or you will get errors when you attempt to compile and deploy your applications). Copy the CSS stylesheet ThreeDCube1.css into your project and merge the relevant portion of the HTML Web page ThreeDCube1.html with the contents of index.html in your project.

### NOTE

*If you are using Xcode 4.x with a PhoneGap plugin from an earlier version of PhoneGap, then you need to perform a manual copy of the generated www subdirectory into the project home directory of your current Xcode application. When you have performed this step correctly you will no longer see an error message when you launch your mobile application in the Simulator or on your iOS device.*

Listing 10.8 displays the contents of the HTML Web page ThreeDCube1.html that references a CSS stylesheet that creates animation effects.

### LISTING 10.8 ThreeDCube1.html

```
<!DOCTYPE html>
<!--
  Licensed to the Apache Software Foundation (ASF) under one
  or more contributor license agreements. See the NOTICE file
  distributed with this work for additional information
  regarding copyright ownership. The ASF licenses this file
  to you under the Apache License, Version 2.0 (the
  "License"); you may not use this file except in compliance
  with the License. You may obtain a copy of the License at

  http://www.apache.org/licenses/LICENSE-2.0

  Unless required by applicable law or agreed to in writing,
  software distributed under the License is distributed on an
  "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY
  KIND, either express or implied. See the License for the
  specific language governing permissions and limitations
  under the License.
-->
<html>
  <head>
    <meta charset="utf-8" />
    <meta name="format-detection" content="telephone=no" />
    <meta name="viewport" content="user-scalable=no, initial-
      scale=1, maximum-scale=1, minimum-scale=1,
      width=device-width, height=device-height,
      target-densitydpi=device-dpi" />

    <!-- commented out to prevent logo.png from being displayed:
    <link rel="stylesheet" type="text/css" href="css/index.css" />
    -->

    <link href="ThreeDCube1.css" rel="stylesheet" type="text/css">
```

```

        <title>CSS 3D Cube Example</title>
    </head>

    <body>
        <div id="outer">
            <div id="top">Text1</div>
            <div id="left">Text2</div>
            <div id="right">Text3</div>
        </div>

        <div class="app">
<!-- commented out to remove default PhoneGap text messages:
        <h1>PhoneGap</h1>
        <div id="deviceready" class="blink">
            <p class="event listening">Connecting to Device</p>
            <p class="event received">Device is Ready</p>
        </div>
-->
        </div>

        <script type="text/javascript" src="phonegap.js"></script>
        <script type="text/javascript" src="js/index.js"></script>
        <script type="text/javascript">
            app.initialize();
        </script>
    </body>
</html>

```

Listing 10.8 is the result of merging the HTML Web page `ThreeDCube1.html` with the HTML Web page `index.html` that is automatically generated by PhoneGap. The first part of Listing 10.8 contains a copyright notice, followed by boilerplate HTML markup, and then a `<link>` element that references the CSS stylesheet `ThreeDCube1.css` (shown in Listing 5.8). The next portion of code contains a `<body>` element followed by some markup that originates from the Web page `ThreeDCube1.html`. The remaining portion of Listing 5.7 contains autogenerated code that references the appropriate JavaScript files to perform PhoneGap-related initialization.

Listing 10.9 displays a portion of the contents of the CSS stylesheet `ThreeDCube1.css` that contains CSS selectors for creating animation effects. The entire contents of `ThreeDCube1.css` are available on the CD.



#### **LISTING 10.9** *ThreeDCube1.css*

```

@-webkit-keyframes animCube1 {
    0% {
        -webkit-transform: matrix(1.5, 0.5, 0.0, 1.5, 0, 0)
                           matrix(1.0, 0.0, 1.0, 1.0, 0, 0);
    }

    10% {
        -webkit-transform: translate3d(50px,50px,50px)
                           rotate3d(50px,50px,50px,-90deg) skew
                           (-15deg,0) scale3d(1.25, 1.25, 1.25);
    }
}

```

```

    }
    // details omitted for brevity
}

#outer {
  -webkit-animation-name: animCube1;
  -webkit-animation-duration: 40s;
}
// more details omitted for brevity

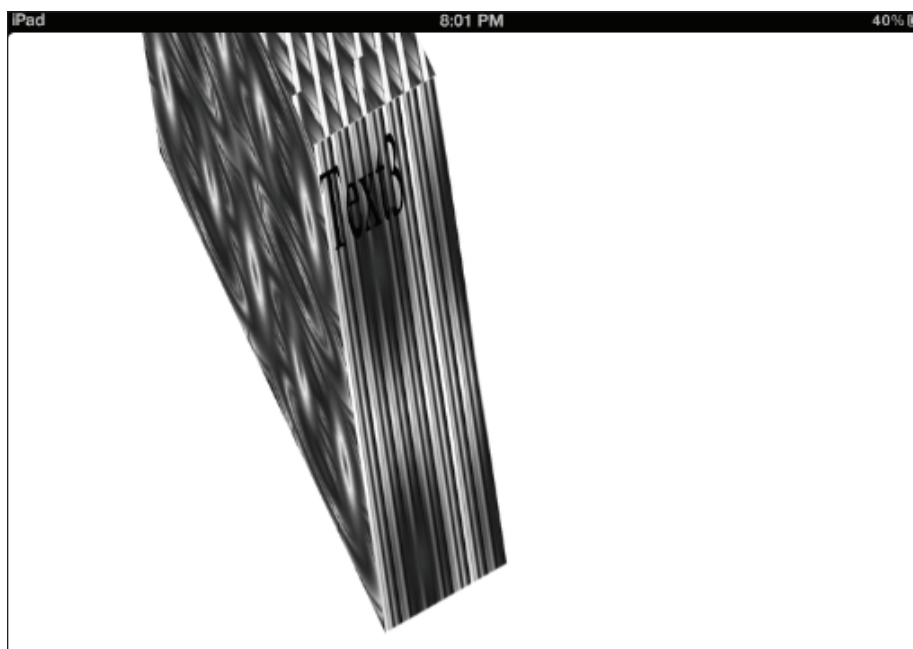
```

Notice that Listing 10.9 defines a CSS3 `keyframes` rule called `animCube1` that is referenced in the selector `#outer`, which in turn matches an HTML `<div>` element in Listing 10.8 that encloses the 3D cube.

Run this mobile application, either in the Xcode Simulator or on your mobile device, and you will see a graphics image that is similar to Figure 10.5.

The process for creating the other iOS-based mobile applications in this chapter is identical to the process for the preceding iOS mobile application, so there is no need to include additional examples. However, it's worth your while to spend some time creating additional iOS mobile applications, which will increase your comfort level and perhaps motivate you to learn about other features of Xcode.

This concludes the brief introduction to generating hybrid mobile applications using version 3.0 of PhoneGap, and you can find additional information about PhoneGap 3.0 at <http://docs.phonegap.com/en/3.0.0/index.html>.



**FIGURE 10.5** A CSS3 Cube on an iPad3.

## D3 AND ANDROID APPLICATIONS

Launch Eclipse and create an Android project called `D3SimpleShapes1`; make sure that you select Android version 3.1 or higher, which is necessary to render SVG elements.

Using `$APP_TOP` to indicate the top-level directory of the Android mobile application in this section, the three files of interest are here:

- `$APP_TOP/src/com/example/d3simpleshapes1/D3SimpleShapes1.java`
- `$APP_TOP/assets/www/SimpleShapes1.html`
- `$APP_TOP/res/layout/activity_main.xml`

Listing 10.10 displays the contents of the Java class `D3SimpleShapes1.java` that launches the HTML5 Web page `D3SimpleShapes1.html`.

**NOTE** *The HTML5 Web page is a copy of `SimpleShapes1.html` that is discussed in Chapter 1, so there is no need to discuss its contents in this section.*

Listing 10.10 displays the contents of `activity_main.xml` that contains an Android `WebView` that is required to launch HTML5 Web pages in Android.

### **LISTING 10.10** *D3SimpleShapes1.java*

```
package com.example.d3simpleshapes1;

import android.app.Activity;
import android.os.Bundle;
import android.view.KeyEvent;
import android.view.Menu;
import android.webkit.WebView;
import android.webkit.WebViewClient;

public class D3SimpleShapes1 extends Activity
{
    WebView mWebView;

    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        mWebView = (WebView) findViewById(R.id.webview);
        mWebView.setWebViewClient(new ChangeURLClient());
        mWebView.getSettings().setJavaScriptEnabled(true);
        mWebView.getSettings().setDomStorageEnabled(true);
        mWebView.loadUrl("file:///android_asset/www/SimpleShapes1
                        .html");
    }
}
```



```

@Override
public boolean onCreateOptionsMenu(Menu menu) {
    // Inflate the menu; this adds items
    // to the action bar if it is present
    getMenuInflater().inflate(R.menu.main, menu);
    return true;
}

public boolean onKeyDown(int keyCode, KeyEvent event) {
    if ((keyCode == KeyEvent.KEYCODE_BACK) &&
        mWebView.canGoBack()) {
        mWebView.goBack();
        return true;
    }

    return super.onKeyDown(keyCode, event);
}
}

```

Listing 10.10 starts with various import statements, and after declaring the Java class `D3SimpleShapes1`, the code contains the definition of the variable `mWebView`:

```
WebView mWebView;.
```

The main focus of attention is the contents of the `onCreate()` method that initialize `mWebView` and various properties, as shown here:

```

mWebView = (WebView) findViewById(R.id.webview);
mWebView.setWebViewClient(new ChangeURLClient());
mWebView.getSettings().setJavaScriptEnabled(true);
mWebView.getSettings().setDomStorageEnabled(true);

mWebView.loadUrl("file:///android_asset/www/SimpleShapes1.html");

```

The preceding code snippet (shown in bold) references the HTML5 Web page `SimpleShapes1.html` located in the directory `$APP_TOP/assets/www`.

**NOTE** *You must manually create the directory `$APP_TOP/assets/www` and place a copy of `SimpleShapes1.html` and `d3.min.js` into this directory.*

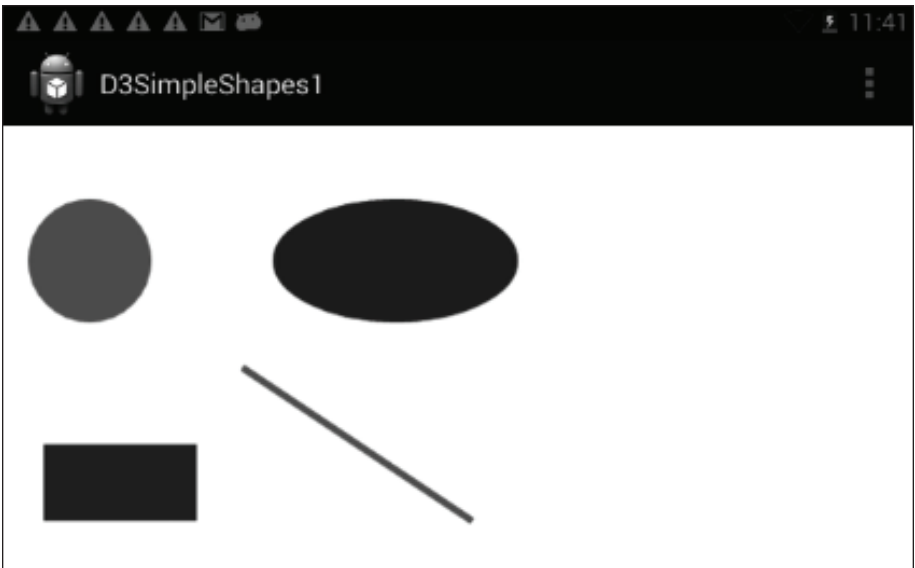
#### **LISTING 10.11** *activity\_main.xml*

```

<WebView
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/webview"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
/>

```

The contents of Listing 10.6 are straightforward: there is one XML `<WebView>` element that is used for rendering the graphics in Listing 10.5.



**FIGURE 10.6** An HTML5 Mobile Application with D3 on Android.

Figure 10.6 displays the result of launching Listing 10.5 on an Asus Prime 10-inch Tablet with Android 4.0.3.

## D3 AND IOS APPLICATIONS

The iOS mobile application in this section is based on the HTML5 Web page `Multi3DBarChart1.html` that is discussed in Chapter 3, so its contents are not duplicated here. The focus of this section is to show you the steps that are required for setting up the iOS mobile application. Another point to remember is that the D3 code was tweaked slightly to ensure that the bar chart fits in a screen whose dimensions are 320dp x 480dp.

Launch Xcode, and after selecting the Apache Cordova plugin, create a project called `D3IOSMulti3DBarChart1`. If you choose another name you need to make the corresponding adjustments to the instructions in this section. Next, copy the contents of the `www` subdirectory into the subdirectory `D3IOSMulti3DBarChart1`.

At this point you need to do two more things:

- copy `d3.min.js` into the `www` subdirectory
- merge or reference `Multi3DBarChart1.html` in `index.html`

The merging process can be performed in three steps as discussed here.

First, insert this code snippet after the `<title>` element in `index.html`:

```
<script src="d3.min.js"></script>
```

Second, copy the contents of the `<script>` element in `Multi3DBarChart1.html` into the following JavaScript method in `index.html` (and also delete the existing pair of lines):

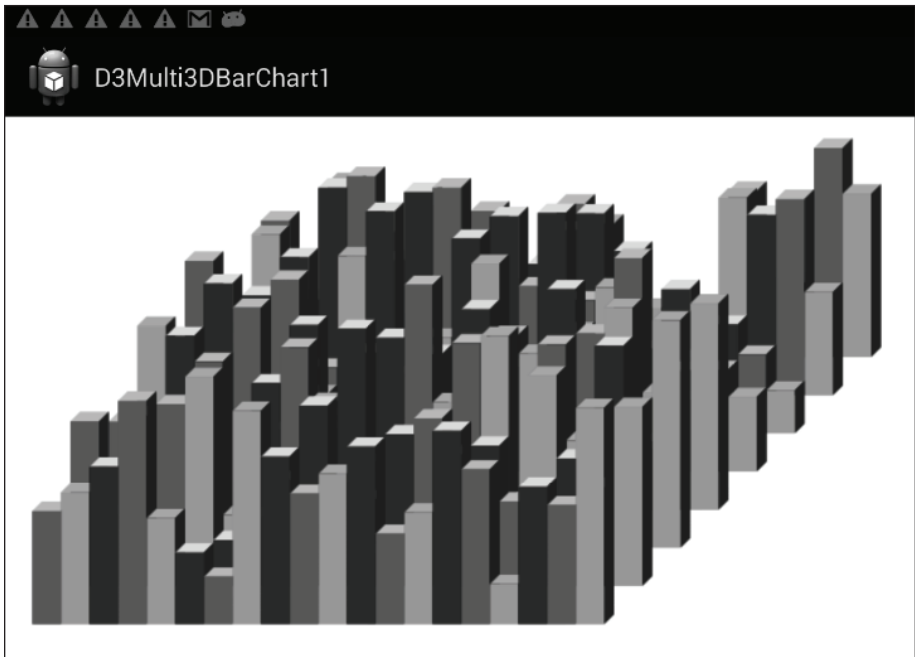
```
function onDeviceReady()
{
    // do your thing!
    navigator.notification.alert("PhoneGap is working citrullo!")
}
```

Third, remove the boilerplate code in the `<body>` element of `index.html`.

At this point, you can launch this mobile application in the simulator or deploy it to a mobile device.

Figure 10.7 displays the result of launching this iOS mobile application on an iPad3 and then tapping on the first several rows of bar elements, which creates a scaled-down bar chart in the upper left corner of the screen in addition to the main bar chart.

This concludes the introduction to PhoneGap. The next portion of this chapter provides an overview of Google Glass, which is an interesting (and perhaps even controversial) Google device.



**FIGURE 10.7** A D3 Bar Chart in an iOS Mobile Application on an iPad3.

## DEVELOPING D3-BASED MOBILE APPLICATIONS FOR GOOGLE GLASS

---

Google Glass is a portable hands-free wearable device that supports Android 4. Google Glass enables you to issue voice commands to take pictures (simply say “take a picture”), get directions, ask for information (“how long is the Brooklyn bridge?”), and even translate your voice into a different language (“say ‘half a pound’ in Chinese”). As you can imagine, the ability to issue voice commands enables you to capture spontaneous events that might otherwise be lost because they only lasted for a few seconds, and you didn’t have a camera or enough time to take a picture or video.

When tethered to your phone, Google Glass can also make calls and send text messages. You can also use Google Glass with an Android phone to screen-cast what is currently on display in Google Glass.

As this book goes to print, Google Glass will use OLED (organic light-emitting diode) displays from Samsung. OLED is transparent, flexible, and almost unbreakable, which will make Google Glass a very durable product.

As a developer, you can interact with a user’s timeline by invoking the appropriate RESTful endpoint to perform the desired action. Google handles the necessary details of synchronizing between your Glassware and your users’ Glass. Some common actions that you can perform include:

- creating and managing timeline cards on a user’s Glass
- subscribing to notifications from Glass to be notified of user actions
- obtaining a user’s location

### How does Google Glass Work?

---

Google Glass provides software called Glassware that enables you to develop Glass mobile applications. Google Glass supports the Google Mirror API, which is a set of RESTful services that transmit information to and receive notifications from Glass devices. The Google Mirror API Playground is available for experimenting with how content is displayed on Glass at <https://developers.google.com/glass/playground>.

Google Glass uses a timeline that contains so-called cards, each of which is used to display information to users. Users navigate through their own timeline by swiping backward and forward on Glass, thereby displaying cards in the past and future.

Each timeline card contains information pushed to Glass devices from various pieces of Glassware. In addition, there are default timeline cards that are pinned to a timeline, so they always appear in the same place. The card that displays the current time and the card that displays all of the tasks that Glass can execute are examples of pinned cards.

Many timeline cards have additional interactions associated with them that are accessible with a single tap. You can define these menu items to allow users to execute actions such as deleting or sharing a card.

Currently, Google Glass does not provide a browser, which means that you cannot deploy and render HTML5 Web pages containing JavaScript, SVG, or HTML5 `<canvas>` elements. However, the good news is that you can deploy Android applications to Google Glass if they are rendered in a `WebView` component. Consequently, you can deploy the hybrid HTML5 Web applications in this book to Google Glass. The details of deploying to Google Glass (which involves the Android `adb` utility) are discussed later in this chapter.

Before discussing the deployment of Android mobile applications to Google Glass, let's take a quick look at the set of HTML5 tags that are supported (and not supported) in Google Glass applications, which are listed in the next two sections.

## Supported HTML5 Tags

There are various allowed HTML elements when using the Glass Mirror API. The allowed HTML elements are elements that you can use in timeline cards. Allowed header tags are `h1`, `h2`, `h3`, `h4`, `h5`, and `h6`. Allowed image tags are `img`. Allowed list tags are `li`, `ol`, and `ul`. Allowed HTML5 semantic tags are `article`, `aside`, `details`, `figure`, `figcaption`, `footer`, `header`, `nav`, `section`, `summary`, and `time`. Allowed structural tags are `blockquote`, `br`, `div`, `hr`, `p`, and `span`. Allowed style tags are `b`, `big`, `center`, `em`, `i`, `u`, `s`, `small`, `strike`, `strong`, `style`, `sub`, and `sup`. Allowed table-related tags are `table`, `tbody`, `td`, `tfoot`, `th`, `thead`, and `tr`.

## Unsupported HTML5 Tags

Blocked HTML elements (and their contents) are removed from HTML payloads. Blocked document headers are `head`, `title`. Embeds: `audio`, `embed`, `object`, `source`, and `video`. Blocked frames tags are `frame` and `frameset`. Blocked scripting tags are `applet` and `script`. Any elements not listed above are removed, but their contents are preserved.

## Deploying Android Applications to Google Glass

The good news is that none of the preceding restrictions for HTML elements are applicable when you launch a `WebView` in a native Android application. You can deploy Android applications to Google Glass using the `adb` command from the command line in the same way that you can deploy Android applications to other Android mobile devices via the command line.

For example, you can deploy the Android application `ThreeDCube1.apk` (that is available on the CD) to Google Glass by performing the following steps:



- connect a USB cable from your machine to Google Glass
- navigate to the directory with the Android `apk` file
- run the command: `adb install ThreeDCube1.apk`

After completing the preceding steps, you launch the Android `apk` file via the `adb` utility from the command line by specifying the fully qualified



**FIGURE 10.8** A Snapshot of an Animated CSS3 3D Cube on Google Glass.

package name of the Android activity and the name of the Activity. In the case of the Android Web application `ThreeDCube1`, the package name is `com.iquarkt.graphics`, and the name of the Android Activity is `ThreeDCube1Activity`, so the command looks like this:

```
adb shell am start -a android.intent.action.MAIN -n com.iquarkt.
css3/.ThreeDCube1Activity
```

After issuing the preceding command, tap the Google Glass, and after a few moments you ought to see the flying 3D cube appear in Google Glass. Listing 10.8 displays a snapshot of a pure-CSS3 cube with 3D animation effects.

### Displaying Google Glass in an Emulator

You can also display whatever is being rendered in Google Glass in an emulator that you can launch from the command line. You need to launch the JAR (Java ARchive) file whose name starts with `asm` (followed by an Android version number) that is located in the directory `$ANDROID_TOP/tools/lib`. For example, the command that you need to invoke for Android 4.0.x is shown here:

```
java -jar $ANDROID_TOP/tools/lib/asm-4.0.jar
```

Launch the preceding command so you can view the rendering of the 3D cube (in the previous section) at the same time that it is being rendered on Google Glass.

One point to keep in mind is that there is a time delay on the emulator, and the transitions are not quite as smooth as they are in Google Glass.

## Other Useful Links for Google Glass

The PlayGround site to test how an HTML card is rendered can be found at <https://developers.google.com/glass/playground>.

If you want to see a fully deployed version of the starter project to get an idea of how it works before you start your own development, navigate to <https://glass-java-starter-demo.appspot.com/>. If you are planning to develop Google-Glass applications, make sure that you read the developer policies found at <https://developers.google.com/glass/policies>.

Various Google Glass starter projects (available in Java, Python, and PHP) are available at <https://developers.google.com/glass/downloads/>.

## Other Google Glass Code Samples

Lance Nanek has created some interesting Google-Glass applications that you can view on github at <https://github.com/lnanek>.

This concludes our coverage of Google Glass in this chapter. Although Google Glass is still in its infancy, it has tremendous potential in terms of the set of applications that developers can create for this mobile device.

## ADDITIONAL CODE SAMPLES ON THE CD



The Android project `HTML5CanvasBBall12` contains the HTML5 Web page `HTML5CanvasBBall12.html` that contains JavaScript code for creating a bouncing ball effect in HTML5 Canvas.

The Android project `PhoneGapForm1` contains the HTML5 Web page `PhoneGapForm1.html` (which will actually be named `index.html` in your Android project) that illustrates how to create a form for various types of user input in Apache Cordova. The types of the input fields are such that the following occurs when users navigate to this form:

- text input displays a standard keyboard
- telephone input displays a telephone keypad
- URL input displays a URL keyboard
- email input displays an email keyboard
- zipcode input displays a numeric keyboard

## SUMMARY

This chapter showed you how to create hybrid Android mobile applications that contain HTML5, CSS3, and SVG. You learned how to create such mobile applications manually, which involved creating Android projects in Eclipse and then modifying the contents of the Android `Activity` class and populating an assets subdirectory with HTML-related files.

Next, you learned how to use PhoneGap 3.0, which simplifies the process of creating mobile applications. Finally, you learned how to deploy Android apk files to Google Glass and how to launch those apk files from the command line.

# INDEX

## A

- Ajax requests, 173–177
- Android applications
  - D3, 237–239
  - Google Glass, 242–243
  - HTML5 Canvas, 223–225
  - HTML5 Canvas multiline graphs, 225–228, 229
  - HTML5/CSS3, 218–221
  - SVG, 221–223
- AngularJS, 211
- animation effects, 41–43, 214–215
  - with bar charts, scrolling, 72–73
  - in CSS3, 158–160
    - three-dimensional, 148–151
  - histogram with, 89–91
  - SVG, 121
- Apache Cordova, 217, 218, 239, 244
- arrays
  - maximum and minimum values in, finding, 26
  - multidimensional, 26–28
  - of numbers to different ranges, scaling, 14–16
  - two-dimensional, 29–30
- attribute selection, in CSS3, 130–131

## B

- BackboneJS, 211
- bar charts, 47–75
  - date and time stamps to label axes, using, 54–55
  - dynamically adding and removing data from, 71–72
  - with filter effects, 61–67
  - horizontal and vertical axes with labels, rendering, 49–52
  - horizontal bar chart, 47–49
  - scrolling animation effects with, 72–73
  - with three-dimensional effects, 57–61
    - from CSV files, 99–101
  - with tooltips, 61–63
  - with Unicode characters, 55–56
  - updating, 67–71
  - with WebSocket server, 180–187
- Bezier curves
  - SVG, 110–114
  - and text, 12–14
- binding data to document object model elements, 6–7



- blocks, 216
- boilerplate, 2–3
- Bostock, Mike, 1
- box shadow effects, in CSS3, 135
- bubble chart with JSON data, 168–170, 171

## C

- CartoDB, 199
- cascading style sheets (CSS), 51, 52
  - D3 force with, using, 203
- CCS. *See* cascading style sheets (CSS)
- ChartBuilder extension, 209
- Choropleth map, 193–194
- comma-separated values (CSV) files
  - bar charts with three-dimensional effects from, 99–101
  - D3 methods for working with, 93
    - synchronous versus nonsynchronous, 94
  - data, line graphs with, 94–98
- common idiom, 5–6
- CrossFilter extension, 209, 214
- CSG, 215
- CSS. *See* cascading style sheets (CSS)
- CSS3, 127–154
  - attribute selection, 130–131
  - based animation effects with D3, 158–160
  - based graphics effects with D3, 156–158
  - browser-specific prefixes for, 128
  - cube on iOS using PhoneGap, 234–236
  - features of, 129
  - gradients
    - linear, 137–140
    - radial, 140–142
  - media queries, 151–152
  - pseudoclasses, 130
  - relational symbols, 130
  - rotate transforms, 145–148
  - rounded corners, 135–137

- shadow effects
  - box, 135
  - colors with RGB triples and HSL representation, 131–132
  - text, 132–135
- support, 128
- and SVG
  - bar charts, 124
  - reference documents using selectors, 123, 152–153
  - similarities and differences between, 124
- three-dimensional animation effects, 148–151
- two-dimensional transforms, 142–145
- updated stylesheets, 156
- CSV. *See* comma-separated values (CSV) files
- Cubism, 211–212

## D

- D3 (Data-Driven Documents). *See also individual entries*
  - and Ajax requests, 173–177
  - and Android applications, 237–239
  - animation effects with, 41–43, 214–215
  - application programming interface
    - reference, 21
  - bar charts with WebSocket server, 180–187
  - based graphics and SVG graphics, combining, 164–168
  - boilerplate, 2–3
  - common idiom in, 5–6
  - CSS3-based animation effects with, 158–160
  - CSS3-based graphics effects with, 156–158
  - data scaling functions, 30–33
  - definition of, 1–2
  - easing functions in, 43–44
  - extensions, 209–211

- and external resources, 94
- force layout, 200–202
- force with CSS, using, 203
- and Google Maps, 198
- and HTML5 Canvas code,
  - combining, 155–156, 160–164
- and iOS applications, 239–240
- keyboard events, handling,
  - 44–45
- maps in, 193–197
- method chaining in, 3
- on mobile devices, 2
- and mouse events, 36–41
- with MySQL data, 178–180
- as NodeJS module, 187–191
- path data generator, 33–35
- with PHP data, 177–178
- plugins, 211–212
- selection-based methods, 3–4
- three-dimensional graphics,
  - 214–215
- trees, 203–205
- tweening in, 16–17
- D3 Brushes, 214
- D3.js gallery, 215
- Data-Driven Documents.
  - See* D3
- data files, 92
- data formats, 92
- data scaling functions, 30–33
- date and time stamps to label axes,
  - using, 54–55
- dc.js extension, 209
- DexChart, 213
- document object model (DOM), 5
  - elements, binding data to, 6–7
- DOM. *See* document object model (DOM)
- drag events, handling, 39–41
- DVL. *See* Dynamic Visualization LEGO (DVL)
- Dynamic Visualization LEGO (DVL), 212

## E

- easing functions, 43–44
- Eclipse, 218, 221, 223, 225, 230, 237, 244
- EmberJS, 211
- external resources, 94

## F

- filter effects, bar charts with,
  - 61–67
- filters, SVG, 114–115
- force layout, 200–202

## G

- GeoJSON, 198–199
- Gist, 216
- Google Glass
  - Android applications to,
    - deploying, 242–243
  - code samples, 244
  - D3-based mobile applications for,
    - developing, 241–244
  - in emulator, displaying, 243
  - functionality of, 241–242
  - links associated with, 244
  - supported HTML5 tags, 242
  - unsupported HTML5 tags, 242
- Google Maps, 198
- gradients
  - effects, adding HTML elements
    - with, 19–20
  - linear, 17–18, 137–140
  - radial, 18–19, 140–142
  - SVG, 107–109

## H

- heat map, 199
- height map, 199
- histogram with animation effects, 89–91
- horizontal bar chart, 47–49
  - with labeled axes, 49–52
- HTML5 Canvas
  - Android application of, 223–225
  - multiline graphs, 225–228, 229

HTML5 Canvas code and D3 code,  
combining, 155–156, 160–164

HTML5/CSS3, Android application of,  
218–221

HTML5 Web Audio, 214

HTML5 Web page  
with D3 code, specifying  
UTF-8 in, 4

SVG in, 122

HTML elements

creating, 4–5

with gradient effects, adding, 19–20

## I

iOS

applications

D3, 239–240

PhoneGap 3.0, 233

devices, deploying mobile

applications to,

requirements for, 233

## J

JavaScript

SVG and, 122–123

jQuery .ajax() method, 93

jsFiddle, 216

## K

keyboard events, handling, 44–45

## L

linear gradients, 17–18

CSS3, 137–140

linear scaling functions, 33

line graphs, 77–80

with CSV data and mouse events,  
94–98

## M

maps, 193–199

choropleth, 193

Google Maps, 198

method chaining, 3

mobile devices, D3 on, 2

mouse events, 36–41

creating and rendering circles at  
current location, 38–39

creating and rendering circles at  
random location, 37–38

drag-and-drop, 39–41

line graphs with, 94–98

scatter charts with, 83–86

multidimensional arrays, 26–28

multiple nonlinear graphs, 80–83

MySQL data, D3 with, 178–180

## N

NodeJS module, D3 as, 187–191

numbers

arrays of numbers to different  
ranges, scaling, 14–16

formatting, 17

NVD3, 213

## O

ordinal scaling functions, 33

## P

pan effects, 44

path data generator, 33–35

PhoneGap, 228–233

CSS3 cube on iOS using, 234–236

working of, 228

PhoneGap 2.9, 230

PhoneGap 3.0

Android hybrid applications with,  
creating, 230–232

iOS hybrid applications with,  
creating, 233

software dependencies for, 229–230

PHP data, D3 with, 177–178

pie charts, 88–89

PlayGround, 244

plugins, 211–212

Plunker, 216

PNG. *See* portable network graphics  
(PNG) files

portable network graphics (PNG) files,  
20–21

Protovis, 1

pseudoclasses, in CSS3, 130

## Q

quantitative scaling functions, 33

## R

radial gradients, 18–19

CSS3, 140–142

rescale effects, 44

Rickshaw, 210–211

rotate transforms, in CSS3, 145–148

rounded corners, in CSS3, 135–137

R programming with D3-based  
toolkits, 213

## S

Scalable Vector Graphics (SVG),  
2, 103–126

Android application of, 221–223

animation effects, 121

basic two-dimensional shapes in,  
104–106

Bezier curves, 110–114

and CSS3

bar charts, 124

reference documents using  
selectors, 123, 152–153

similarities and differences  
between, 124

and D3-based graphics, combining,  
164–168

element, 107–109, 109–110, 119–120  
text along, rendering,  
115–116, 117

filters and shadow effects, 114–115

gradients, 107–109

in HTML, 122

in JavaScript, 122–123

in NodeJS, 187–191

three-dimensional effects in,  
121–122

transforms, 110–114, 116–119  
and XSLT, 125

scatter charts, 29–30

with axes and mouse events, 83–86  
equal data points in, selecting,  
87–88

selection-based methods, 3–4

shadow effects

CSS3

box, 135

colors with RGB triples and HSL  
representation, 131–132

text, 132–135

SVG, 114–115

simple two-dimensional shapes,  
creating, 9–11

SVG. *See* Scalable Vector Graphics  
(SVG)

## T

text shadow effects, in CSS3,  
132–135

text strings, generating, 8–9

“this” keyword, 35

three-dimensional bar charts, 57–61  
with CSV data, 99–101

three-dimensional effects  
in SVG, 121–122

three-dimensional graphics, 214–215

Three.js, 215

TopoJSON, 199

tQuery, 215

trees, 203–205

tweening, 16–17

two-dimensional arrays, 29–30

two-dimensional transforms, 14  
in CSS3, 142–145

## U

Unicode characters, bar charts with,  
55–56

UTF-8

in HTML5 Web pages with D3  
code, specifying, 4

## V

Vega, 212–213

vertical bar chart

with labeled axes, 49–52

scaled, 52–53

Virtual Reality Modelling Language

(VRML), 215

Voronoi diagram, 205–208

VRML. *See* Virtual Reality Modelling Language (VRML)

## W

WebGL, 215

WebSocket server, D3 bar charts with, 180–187

## X

X3D, 215

XMLHttpRequest request object, 92–93

XSLT (Extensible Stylesheet Language Transformations)

SVG and, 125

## Z

zoom effects, 44