显示层动画 / 内容层动画 / 3D动画 / 转场动画

Broadview
www.broadview.com.cn

iOS动画

核心技术与案例实战

· 郑微　编著 ·

基于Swift 3.0，舞动酷炫的iOS动画

中国工信出版集团

電子工業出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY
http://www.phei.com.cn

iOS 动画 —— 核心技术与案例实战

郑　微　编著

電子工業出版社
Publishing House of Electronics Industry
北京 · BEIJING

内 容 简 介

目前，APP Store 上的应用已经超过 150 万个，而纵观排名较为靠前的应用，无一例外都有着一个共同的特点，那就是良好的用户体验。动画作为用户体验中最复杂、最绚丽的技术已经备受开发人员和产品设计人员的重视。而如何将炫酷的动画效果快速高效地展现出来已经成为 iOS 开发工程师面临的首要挑战。

本书以“iOS 核心动画架构＋实战代码”的形式阐述如何根据不同的应用场景设计高效、可靠、复杂的动画效果，为读者带来了丰富的实战动画案例，更从动画系统架构的角度阐释动画的原理，因此本书不仅面向读者“授之以鱼”更加“授之以渔”。

图书在版编目（CIP）数据

iOS 动画：核心技术与案例实战 / 郑微编著．—北京：电子工业出版社，2017.2
ISBN 978-7-121-30748-5

Ⅰ．①i… Ⅱ．①郑… Ⅲ．①移动终端－应用程序－程序设计 Ⅳ．①TN929.53

中国版本图书馆 CIP 数据核字（2016）第 316808 号

策划编辑：杨中兴
责任编辑：黄爱萍
印　　刷：三河市华成印务有限公司
装　　订：三河市华成印务有限公司
出版发行：电子工业出版社
　　北京市海淀区万寿路 173 信箱　　邮编：100036
开　　本：720×1000　1/16　印张：13.25　字数：212 千字
版　　次：2017 年 2 月第 1 版
印　　次：2017 年 2 月第 1 次印刷
定　　价：69.00 元

凡所购买电子工业出版社图书有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系，联系及邮购电话：（010）88254888，88258888。

质量投诉请发邮件至 zlts@phei.com.cn，盗版侵权举报请发邮件至 dbqq@phei.com.cn。

本书咨询联系方式：（010）51260888-819，faq@phei.com.cn。

序　言

平时在家我经常会下载一些 iOS 排名比较靠前的应用或者游戏来玩，这些应用或者游戏都有比较显著的特点：界面优美、运行流畅、效果炫酷。这里暂且不去讨论 iOS 的硬件性能和 UI 美工设计是否优美，只是单纯从动画效果的角度看，它们对 iOS 动画效果的应用有着非常精致的把控。

在日常工作中，每开发一款 APP，我们都会绞尽脑汁想让这款应用与众不同。其实不用太过于纠结系统，因为 iOS 的硬件都很棒。也不必太过于纠结美工，相信一个稍微靠谱的美工做出来的 UI 都不会太差。只需要选择和设计一些比较优美的动画，就可以让自己的应用上一个新的台阶。

在工作过程中大家都经历过这三个阶段。第一阶段初入江湖。在这个阶段如果想设计一个比较炫酷的动画效果，要么请教“大神”，要么进行网络搜索，而很少有自己的想法。这主要是因为大家对动画的架构、常用 API、常用效果没有一个全面的认识，这个阶段基本属于代码收集阶段。第二阶段渐入佳境。相信大家在这个阶段都会有一些自己的思想，通过不断的尝试、对 API 不断地调整都能够实现最终想要的效果。总体来说这一阶段属于代码整理阶段。第三阶段登堂入室。需求来了之后在开发人员的大脑中很快被分解为若干子功能，迅速定位子功能需要实现的代码块。通过“搬砖+修改”的模式实现快速开发。这一阶段基本属于代码灵活运用阶段。

如果从零开始一步步完成这样三个阶段，相信大家都能做到，但是这会花费非常多的时间。在工作中我也曾被第一阶段和第二阶段反复困扰过，走了不少弯路，花费了大量的时间和精力，查看了各种官方手册和相关书籍，直到进入第三阶段才体会到“一览众山小”的感觉。所以我很想把 iOS 关于动画的相关知识为大家抽丝剥茧地整理一番，以帮助更多的人花费更少的时间掌握尽可能多的知识。

1．这本书有哪些特点

（1）层次分明

iOS 动画效果非常丰富，本书一共 16 章，根据动画实现方式及效果分为 4 卷。第一卷（第 1～5 章）介绍显示层动画效果，第二卷（第 6～12 章）介绍内容层动画效果，第三卷（第 13～14 章）介绍 3D 动画效果，第四卷（第 15～16 章）介绍转场动画效果。这种划分有利于读者在学习的过程中对所要查找的动画效果快速定位，以及将知识点分类掌握。

第一卷为显示层动画效果，即利用 UIView 图层显示的效果实现各种动画。常见的有位置动画、颜色动画、淡入淡出动画、旋转动画、关键帧动画、逐帧动画等，文中针对要显示的动画效果一般采用 3 张图渐进描述，对于复杂的动画效果多采用 6 张图或 9 张图描述动画的渐变过程。其效果分别如图 1～图 4 所示。

图 1　位置动画

图 2　颜色渐变动画

图 3　关键帧动画

图 4　逐帧动画

第二卷为内容层动画效果。内容层动画依赖视图的 Layer 图层，结合常用 Layer 子类，如 CAEmitterCell 粒子动画、CAGradientLayer 扫描动画、CAShapeLayer 图表类动画、CAReplicatorLayer 图层快速复制动画等实现内容层动画展示，如图 5～图 8 所示。

图 5　CAEmitterCell：粒子“鬼火”效果

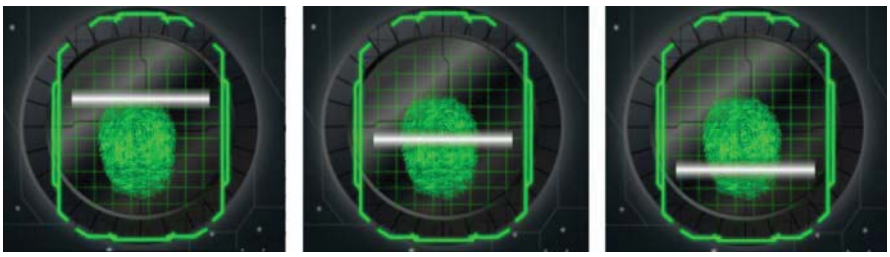


图 6　CAGradientLayer：指纹扫描效果

图 7　CAShapeLayer：动态图表效果

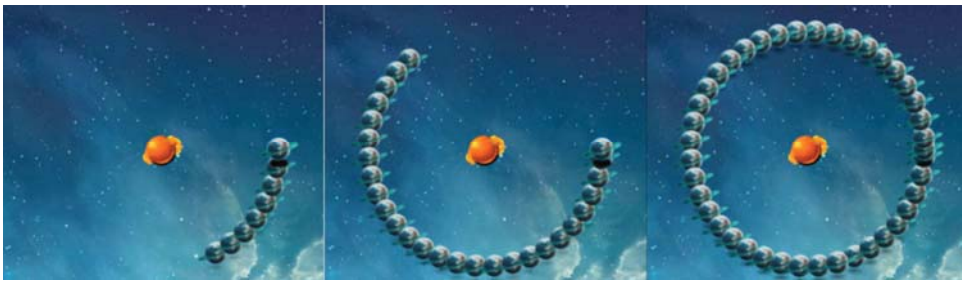


图 8　CAReplicatorLayer：“恒星”公转效果

第三卷为 3D 动画效果。3D 动画效果以矩阵变换为基础，利用 x、y、z 与变换矩阵相互作用实现各种 3D 效果，如图 9 所示。比如 Cover Flow 的 3D 动画展示效果，如图 10 所示。

$$\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} * \begin{bmatrix} m11 & m12 & m13 & m14 \\ m21 & m22 & m23 & m24 \\ m31 & m32 & m33 & m34 \\ m41 & m42 & m43 & m44 \end{bmatrix} = \begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix}$$

coordinate　　transform　　transformed coordinate

图 9　变换矩阵原理解析

图 10　Cover Flow 展示效果

第四卷为转场动画效果。转场动画常用于多视图场景下视图切换，如常见的水滴、翻页、波纹效果，或者自定义视图控制器转场动画，如图 11～图 12 所示。

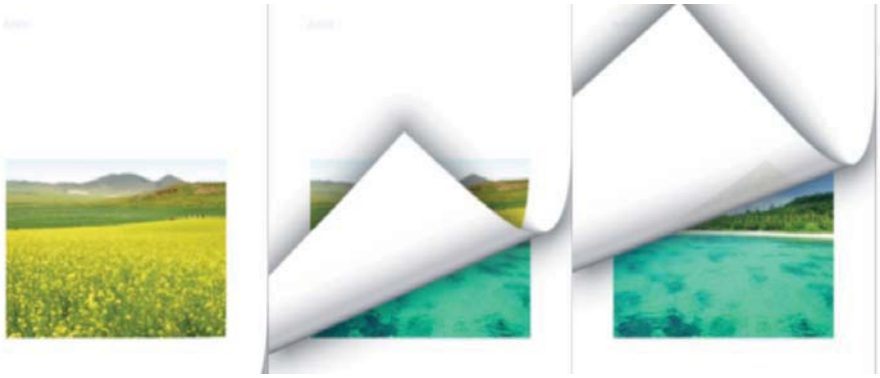


图 11　转场翻页效果

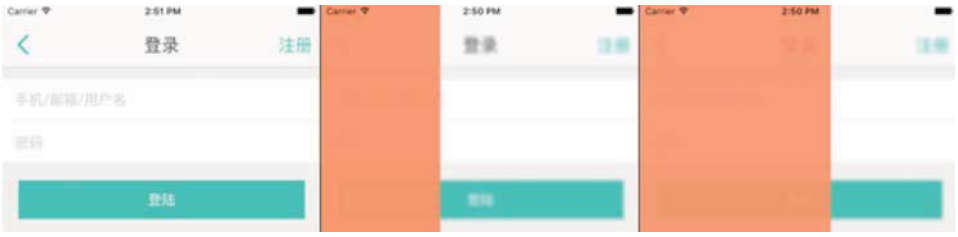


图 12　自定义转场蒙版效果

（2）内容丰富

对于相同类型、相同知识点的动画，书中做了详细的归纳和总结，这种归纳和总结有利于读者对动画整体架构的把握和快速精准的使用，如图 13～图 15 所示是部分动画合集知识点。

图 13　UIView 常用动画合集

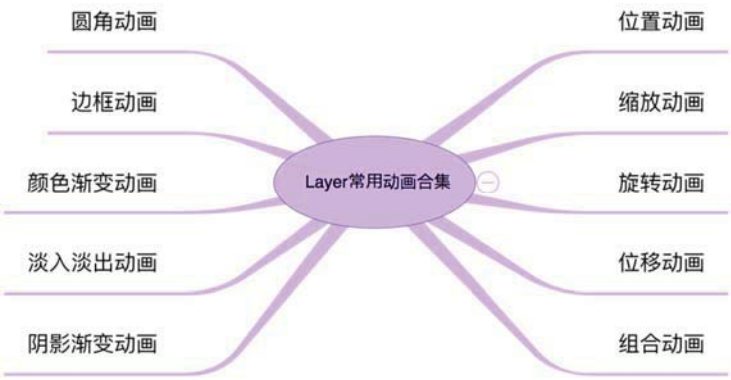


图 14　Layer 常用动画合集

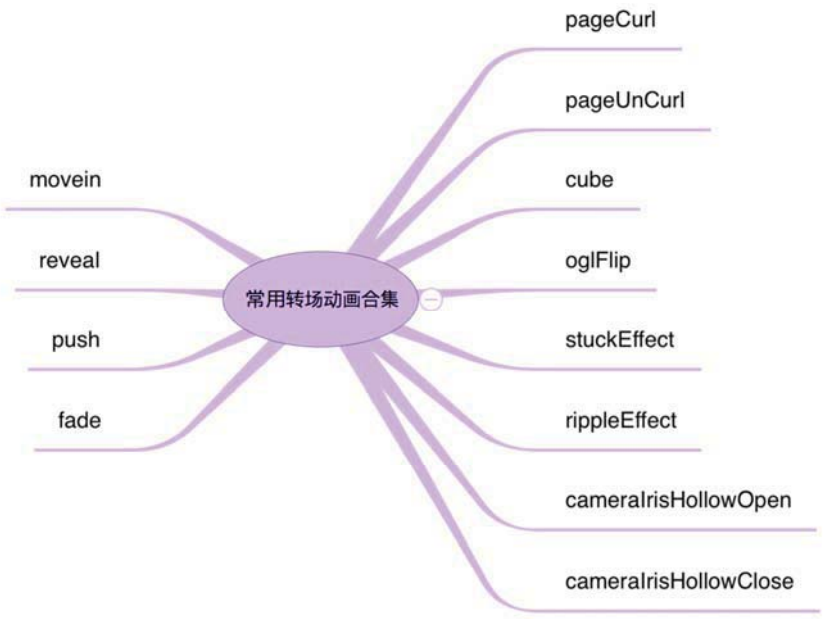


图 15　常用转场动画合集

（3）适用性强

本书针对每个章节给出适合阅读的人群，便于读者过滤出适合自己的核心内容。

（4）实用性强

在讲解每个动画案例的同时，尽可能贴近实际使用场景，如第二卷的各种 Layer 层动画实战案例、Button 按钮相关动画效果等，如图 16～图 17 所示。

图 16　按钮水纹点击效果

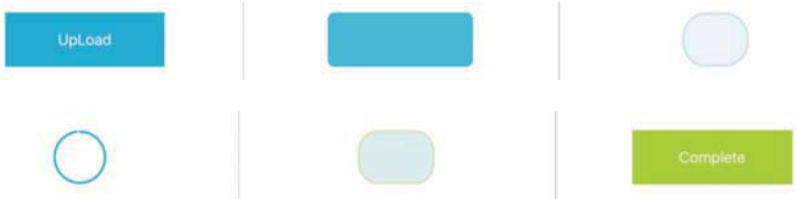


图 17　按钮登录效果

2．iOS 动画架构一览

根据不同维度对动画架构进行划分，可以很好地帮助大家理解 iOS 动画的结构及不同类型动画之间的相互联系。如图 18～图 20 所示。

图 18　iOS 动画分层结构

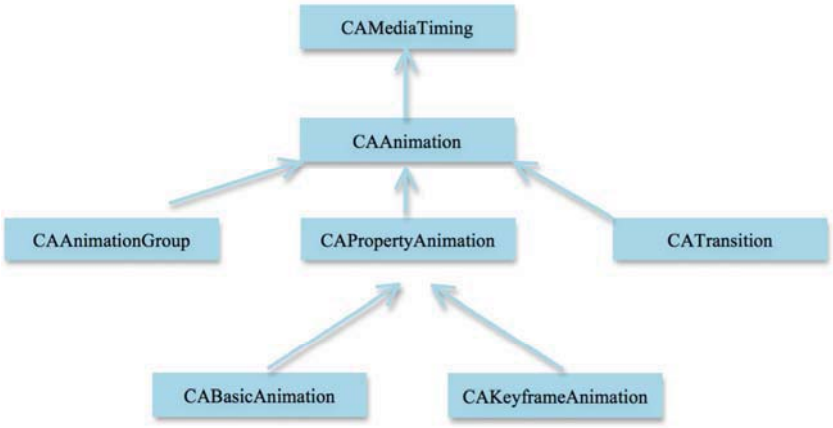


图 19　核心动画类结构

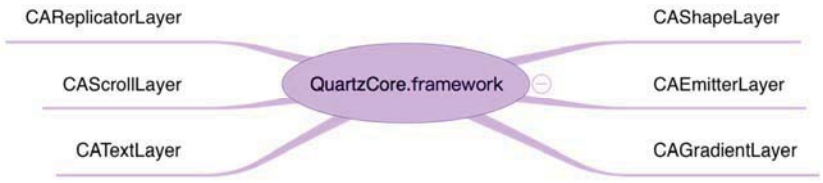


图 20　QuartzCore 框架下常用图层

3．如何使用这本书

俗语说得好：“工欲善其事，必先利其器”，要想很好地使用这本书，必

须先准备好一定的工具。本书中的所有代码都是基于 Swift 语言开发的，建议使用 Xcode 8.0、SDK 10.0 以上版本调试。源码下载地址详见：http://www.broadview.com.cn/30748。

工具和源码准备好之后呢？大家都知道几乎没有一本书百分之百适合我们，但或许有某个章节或者某个知识点刚好是大家想要学习和掌握的，本书也不例外。为了方便大家更好地使用这本书，特将本书的特点描述如下，希望广大读者根据自己的实际情况进行相关内容的阅读学习。

- **iOS 初级开发工程师**：建议从第 1 章动画入门开始，循序渐进地阅读。
- **iOS 中级开发工程师**：建议浏览或者跳过第一卷，重点关注第二、三、四卷。
- **iOS 高级开发工程师**：建议挑选工作中需要或感兴趣的章节阅读，如第 9 章“粒子动画”、第 10 章“光波扫描动画”。
- **iOS 超级开发工程师**：建议从整体架构和动画整理归纳的角度阅读本书。

致谢

在此感谢电子工业出版社的杨中兴编辑为本书提出的宝贵意见，感谢各位技术博主对本书的大力支持。最后感谢一直深爱并默默支持我的妻子，感谢她对我的关心和照顾，使得我可以抽出更多时间全身心地编写此书。

由于时间仓促，书中难免存在不足之处，欢迎大家批评指证。

郑　微

2016 年 12 月于武汉

目　录

第一卷　显示层动画

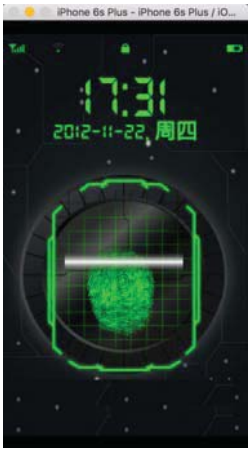
第二卷　内容层动画

第 10 章　CoreAnimation: CAGradientLayer 光波扫描动画效果

本章内容

- **掌握光波扫描“梯度”的基本原理**
- **实现指纹“光波扫描”效果、音响音量跳动效果**

我们在观看科幻大片或者开启重磅音响时，经常会看到如图 10.1 所示的“光波扫描”和如图 10.2 所示的“音量跳动”动画效果，每次看到这些炫酷的动画效果都让人激动不已。本章将使用 QuartzCore 框架下的 CAGradientLayer 类揭开“指纹光波扫描”动画和“音量跳动动画”的神秘面纱。



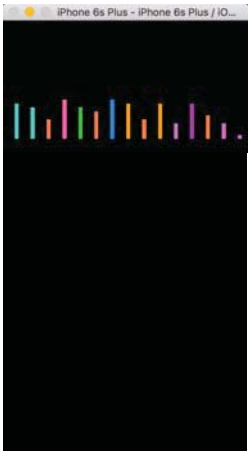


图 10.1　光波扫描效果　　图 10.2　音量跳动效果

10.1　CAGradientLayer 追本溯源

先来追本溯源一下 CAGradientLayer 这个单词。CAGradientLayer 可以拆解为三个主要组成部分：CA、Gradient、Layer。CA 为 CoreAnimation 的缩写，表明当前的动画使用的是 iOS 框架下的核心动画部分。Gradient 为梯度的意思，描述了当前动画的特点，实现一些梯度功能的动画效果。比如位置的梯度变化、颜色的梯度渐变等。Layer 表明当前动画并非直接作用于 UIView 显示层上，而是作用在 Layer 内容层。到这里大家就基本搞清楚了 CAGradientLayer 类的基本含义，接下来想一想如何实现这一动画效果。

相信读者已经阅读了本书的第 1 章内容，具备了动画分析的基本思想。按照第 1 章为大家总结的动画分析方法开始分析。要想利用动画实现的三步曲完成动画的设计，首先要搞清楚动画的算法原理。下面对动画进行逐帧分解，如图 10.3 所示为动画分解效果。

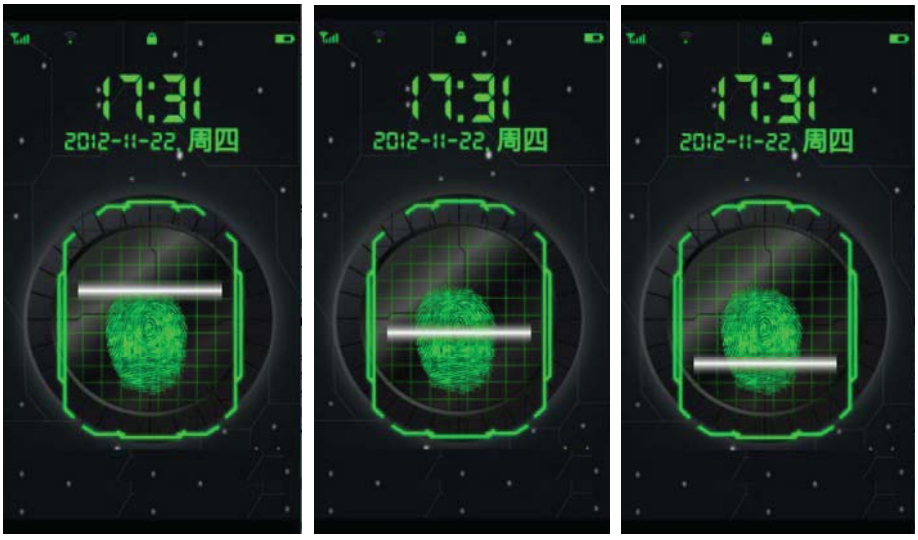


图 10.3　指纹扫描动画多帧分解效果

10.2 光波效果实现原理分析

通过对动画的分解，基本上可以弄清楚动画的演变过程。接下来还需要弄清楚以下几个主要问题。

（1）光波的执行方向。

（2）光波的颜色梯度。

（3）光波的“彗星拖尾”效果。

只有先把这些问题研究透彻了，才能更好地实现图 10.1 所示的指纹扫描效果。下面就从“原理+代码”的角度来阐释这些概念的具体含义。

10.2.1 光波方向

如图 10.4 所示，左图为光波从上到下执行顺序的效果图，右图为光波从左到右执行顺序的效果图。

图 10.4 光波方向

效果已经为大家展示清楚了，那么接下来思考如何通过代码实现这种效果。下面是光波方向设置的核心代码。gradientLayer 为实例化的 CAGradientLayer 实例对象。将 startPoint 设置为（0，0），endPoint 设置为（1，0）即可实现光波从上到下的扫描效果。

```
//      设置光波起始位置和终止位置坐标
        gradientLayer.startPoint = CGPoint(x:0,y:0)
        gradientLayer.endPoint = CGPoint(x:1,y:0)
```

接下来再为大家画一张原理图来分析 startPoint 和 endPoint 的原理。在实例化 CAGradientLayer 时需要设置其 frame。而 startPoint 和 endPoint 与当前 frame 存在一定的映射关系，如图 10.5 所示为二者之间坐标系的映射关系。

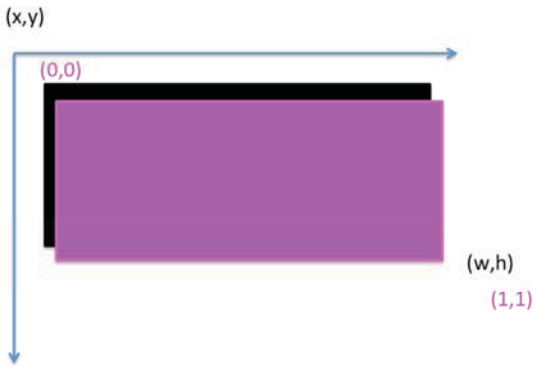


图 10.5　frame 坐标系与“梯度”坐标系对应关系

图 10.5 为 frame 坐标系与“梯度”坐标系的对应关系。其中浅色部分为 startPoint 和 endPoint 的坐标系，黑色部分为实例对象 gradientLayer 的 frame 坐标系。这里把 startPoint 和 endPoint 的坐标系暂且称之为“梯度坐标系”。梯度坐标系范围在 0～1 之间，startPoint 和 endPoint 设置为（0，0）和（0，1）。如图 10.6 所示为梯度坐标系原理图和效果图。

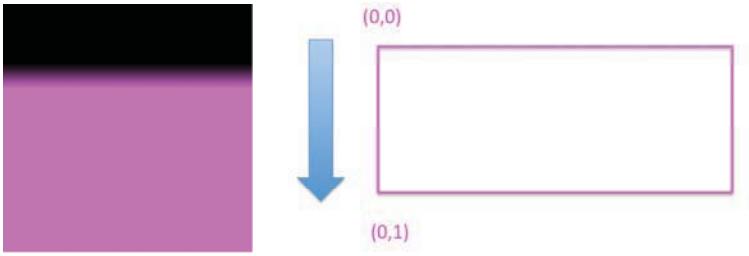


图 10.6　梯度坐标系原理图和效果图

如果想把动画运动方向从上到下改为从左到右，那么只需要将原理图中 startPoint 和 endPoint 分别修改为（0，0）和（1，0）。如图 10.7 所示为水平方向上梯度坐标系原理图和效果图。

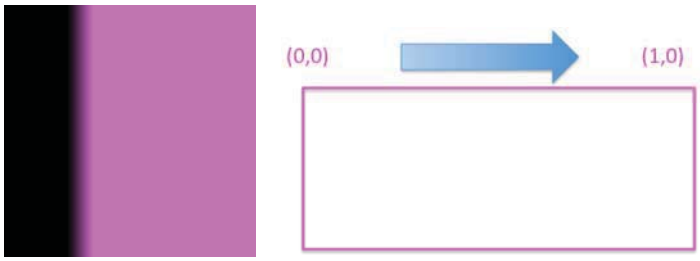


图 10.7　梯度坐标系原理图和效果图

梯度坐标系是一个二维坐标系，除了水平和竖直方向两种情况之外，还有斜线方向。接下来思考一下如何实现斜线方向。如果将 startPoint 设置为（0，0.5），endPoint 设置为（1，0）那么会出现什么效果呢？先来分析原理图，如图 10.8 所示。

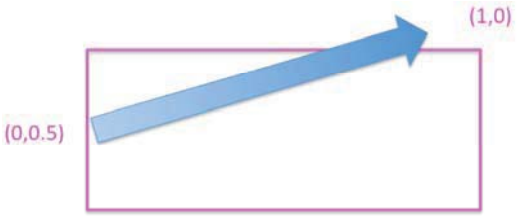


图 10.8　动画斜线方向移动效果图

起始坐标设置为（0，0.5），动画结束坐标设置为（1，0），那么把两点连接起来就是动画移动的起始位置和方向。如图 10.9 所示展示了动画的斜线移动方向。

图 10.9　斜线移动方向动画效果

10.2.2　光波颜色梯度

从图 10.9 中可以看出光波颜色梯度分为两个等级，一个为黑色，另一个为粉红色。下面看看代码如何实现。

```
        let color = UIColor(red: 216.0 / 255.0, green: 114.0 / 255.0,
                            blue: 213.0 / 255.0, alpha: 1.0)
//      设置光波颜色梯度
        gradientLayer.colors = [UIColor.clear.cgColor,
                                color.cgColor]
```

GradientLayer 实例对象的 colors 属性是一个颜色数组，其中数组的第一个元素为 UIView 的底色。当扫描动画扫描完成之后就显示出底色“blackColor”。数组的第二个元素为动画逐渐褪去的颜色。如果设置三种颜色会有什么效果呢？如图 10.10 所示为颜色梯度变化。

图 10.10　颜色梯度变化

可以把代码修改为如下形式来实现图 10.10 的梯度变化效果。

```
        let color = UIColor(red: 216.0 / 255.0, green: 114.0 / 255.0,
                            blue: 213.0 / 255.0, alpha: 1.0) // 粉色
        let color1 = UIColor(red: 61.0 / 255.0, green: 226.0 / 255.0,
                             blue: 210.0 / 255.0, alpha: 1.0)// 青色
//      设置光波颜色梯度
        gradientLayer.colors = [UIColor.clear.cgColor,
                                color1.cgColor,
                                color.cgColor]
```

可以看出颜色数组中第一个颜色为 UIView 最终底色，第二个元素为光波渐变颜色，第三个颜色为原本覆盖在 UIView 上的动画颜色。

10.2.3　光波“彗星拖尾”效果

在正式讲解光波“彗星拖尾”效果之前，先来认识一下 gradientLayer 的颜

色分割属性。这个属性可以很好地帮助我们理解光波“彗星拖尾”效果的原理。把颜色分割属性按照如下代码进行设置。

```
            let color = UIColor(red: 216.0 / 255.0, green: 114.0 / 255.0,
blue: 213.0 / 255.0, alpha: 1.0)// 粉色
            let color1 = UIColor(red: 61.0 / 255.0, green: 226.0 / 255.0,
blue: 210.0 / 255.0, alpha: 1.0)// 青色
    //      设置光波颜色梯度
            gradientLayer.colors = [UIColor.clear.cgColor,
                                    color1.cgColor,
                                    color.cgColor]
            gradientLayer.locations = [0.0,0.1,0.2]
```

最终光波效果如图 10.11 所示。其中 A 到 B 之间的区域为从黑色渐变到青色，B 到 C 之间的区域为从青色渐变到粉色。经过测量，A 与 B 之间的区域大约为整个视图面积的 10%左右，B 与 C 之间的区域面积为整个视图面积的 10%。结合代码：

```
    gradientLayer.locations = [0.0,0.1,0.2]
```

其中 locations 为颜色渐变比例数组。通过图 10.5 可以知道，在“梯度”坐标系中位置坐标的范围在 0～1 之间。而 locations 数组中的三个值分别对应 A、B、C 三个点的位置坐标。由此可知 locations 中 0.0～0.1 描述的即为 A～B 之间的区域，在这片区域中完成了从黑色到青色的颜色渐变过程。locations 中 0.1～0.2 描述的即为 B～C 之间的区域，在这片区域中完成了从青色到粉红色的颜色渐变过程。

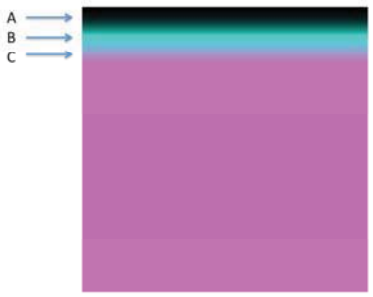


图 10.11　光波颜色“拖尾”效果

在理解了以上的知识之后，再把 A 与 B 之间“彗星拖尾”的效果加深一些。将位置坐标修改为：

```
gradientLayer.locations = [0.0,0.5,0.6]
```

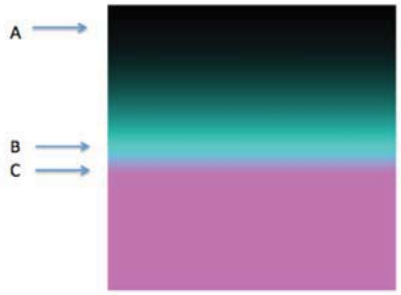


图 10.12　光波颜色“拖尾”增强效果

可见经过修改之后，A 与 B 之间的拖尾效果明显加强了。

10.2.4　光波扫描效果

光波的扫描效果采用 CABasicAnimation 动画来实现。只需要把光波 gradientLayer 的 locations 的属性设置成可变状态就可以实现最终的效果。下面是具体代码实现。

```
1       let gradientAnimation:CABasicAnimation = CABasicAnimation()
2       gradientAnimation.keyPath = "locations"//动画属性
3       gradientAnimation.fromValue = [0.0,0.1,0.2];
//动画属性变化起始状态值
4       gradientAnimation.toValue = [0.8,0.9,1.0];
//动画属性变化终止状态值
5       gradientAnimation.duration = 3.0;//动画执行周期
6       gradientAnimation.repeatCount = 10;//动画执行重复次数
7       gradientLayer.add (gradientAnimation, forKey: nil)
//将基础动画添加到 Layer 图层
```

代码第 1 行初始化 CABasicAnimation 动画实例对象，第 2 行指明当前动画需要实现的动画属性。这里修改的是 locations 属性，而 locations 属性对应的效果如图 10.11 和图 10.12 所示。可见，修改 locations 可以修改光波的位置和拖尾

的效果。第 3 行和第 4 行修改动画初始状态的 locations 值以及动画最终状态 locations 值。第 5 行和第 6 行设置动画执行周期为 3s，第 6 行设置动画重复执行 10 次。最后一行将代码添加到 Layer 图层上，启动当前动画效果。最终的动画效果如图 10.13 所示。

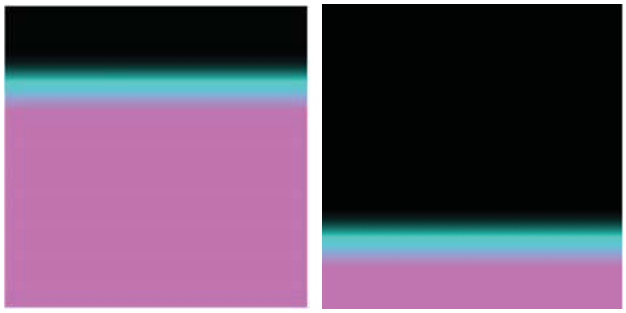


图 10.13　光波扫描效果

10.3　案例：指纹扫描效果

在学习了第 10.2 节光波原理分析之后，相信大家已经有了足够的知识来实现图 10.1 的指纹扫描效果。这里把整个动画效果按功能分解为三个部分（因为这个动画效果比较简单所以没有按照动画的三步曲进行分析，而是根据功能进行划分）。第一部分实现了指纹背景图片的添加，第二部分设置 Layer 图层的相关属性，第三部分利用 CABasicAnimation 实现光波的运动效果。下面是图 10.1 指纹扫描动画效果的全部代码。

```
//      part1:设置指纹扫描背景图片
1       let image:UIImage = UIImage(named: "unLock.jpg")!
2       let imageView:UIImageView = UIImageView(image: image)
3       imageView.contentMode = UIViewContentMode.ScaleAspectFit
4       imageView.frame = self.view.bounds
5       imageView.center = self.view.center
6       self.view.backgroundColor = UIColor.blackColor()
7       self.view.addSubview(imageView)
//      part2:设置 Layer 图层属性
```

```
8       let gradientLayer:CAGradientLayer = CAGradientLayer()
9       gradientLayer.frame = CGRect(x: 105, y: 330,
                                     width: 200,height: 200)
10       imageView.layer.addSublayer(gradientLayer)
11       gradientLayer.startPoint = CGPoint(x:0,y:0)
12       gradientLayer.endPoint = CGPoint(x:,y:1)
13       gradientLayer.colors = [UIColor.clear.cgColor,
                                 UIColor.white.cgColor,
                                 UIColor.clear.cgColor]
14       gradientLayer.locations = [0.0,0.1,0.2]
//     part3:设置 CABasicAnimation
15       let gradientAnimation:CABasicAnimation =
                                          CABasicAnimation()
16       gradientAnimation.keyPath = "locations"
17       gradientAnimation.fromValue = [0.0,0.1,0.2];
18       gradientAnimation.toValue = [0.8,0.9,1.0];
19       gradientAnimation.duration = 3.0;
20       gradientAnimation.repeatCount = 10;
21      gradientLayer.add (gradientAnimation, forKey: nil)
```

代码第一部分：添加指纹背景图片，并设置图片拉伸方式和 frame 等属性添加到 self.view 上。代码第二部分：初始化 Layer 属性，并将 Layer 图层添加到 imageView 的 Layer 图层上。代码第 11 行到第 14 行设置 Layer 图层的相关属性，具体属性设置可以参考第 10.2 节光波原理分析。代码第三部分设置 CABasicAnimation 动画。我们需要修改动画的 locations 属性，并设置动画的起始位置为［0.0，0.1，0.2］，终止位置为［0.8，0.9，1.0］。设置动画执行周期 3s，执行次数 10 次。最后一行将动画添加到 Layer 图层上，启动当前动画。

10.4　案例：音响音量跳动效果

在这个案例中将实现一个类似音量跳动的动画效果，如图 10.14 所示为最终要实现的音量跳动动画效果图。整个动画效果由 15 根柱状 View 组成，并且不同的 View 之间设定了几组随机颜色。

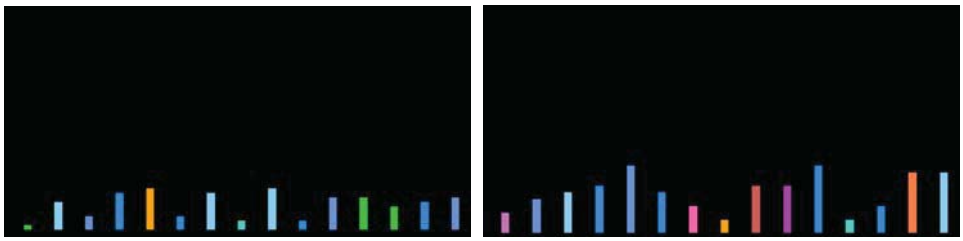


图 10.14　音量跳动动画效果

所以将动画的设计思路总结为以下内容。

（1）动画起始阶段：一组随机颜色。

（2）动画起始阶段：15 根柱状 UIView 视图。

（3）动画进行阶段：如图 10.15 所示。

（4）动画结束阶段。

图 10.15　单根柱状图动画渐变过程

1．动画起始阶段

先来实现动画的起始阶段，将随机颜色和 15 根柱状图绘制到当前屏幕上。下面是准备的 11 组随机颜色代码。通过颜色的 RGB 属性，生成 11 组随机颜色，并将颜色存储到颜色数组 colorArray 中。代码如下所示。

```
func setcolorArray(){
1       colorArray = NSMutableArray()
2       let  color1:UIColor = UIColor(red: 255.0 / 255.0,
```

```
            green: 127.0 / 255.0, blue: 79.0 / 255.0, alpha: 1.0)
3       let  color2:UIColor = UIColor(red: 138.0 / 255.0,
            green: 206.0 / 255.0, blue: 245.0 / 255.0, alpha: 1.0)
4       let  color3:UIColor = UIColor(red: 216.0 / 255.0,
             green: 114.0 / 255.0, blue: 213.0 / 255.0, alpha: 1.0)
5       let  color4:UIColor = UIColor(red: 51.0 / 255.0,
            green: 207.0 / 255.0, blue: 48.0 / 255.0, alpha: 1.0)
6       let  color5:UIColor = UIColor(red: 102.0 / 255.0,
            green: 150.0 / 255.0, blue: 232.0 / 255.0, alpha: 1.0)
7       let  color6:UIColor = UIColor(red: 255.0 / 255.0,
            green: 105.0 / 255.0, blue: 177.0 / 255.0, alpha: 1.0)
8       let  color7:UIColor = UIColor(red: 187.0 / 255.0,
            green: 56.0 / 255.0, blue: 201.0 / 255.0, alpha: 1.0)
9       let  color8:UIColor = UIColor(red: 255.0 / 255.0,
            green: 163.0 / 255.0, blue: 0.0 / 255.0, alpha: 1.0)
10      let  color9:UIColor = UIColor(red: 203.0 / 255.0,
            green: 93.0 / 255.0, blue: 92.0 / 255.0, alpha: 1.0)
11      let  color10:UIColor = UIColor(red: 61.0 / 255.0,
            green: 226.0 / 255.0, blue: 210.0 / 255.0, alpha: 1.0)
12      let  color11:UIColor = UIColor(red: 25.0 / 255.0,
            green: 146.0 / 255.0, blue: 255.0 / 255.0, alpha: 1.0)
13      colorArray.addObject(color1)
14      colorArray.addObject(color2)
15      colorArray.addObject(color3)
16      colorArray.addObject(color4)
17      colorArray.addObject(color5)
18      colorArray.addObject(color6)
19      colorArray.addObject(color7)
20      colorArray.addObject(color8)
21      colorArray.addObject(color9)
22      colorArray.addObject(color10)
23      colorArray.addObject(color11)
    }
```

在 ViewDidload()方法中将所需要的 15 根柱状 View 绘制到当前视图上，代码如下所示。

```
override func viewDidLoad() {
      super.viewDidLoad()
1      setcolorArray()
2      self.view.backgroundColor = UIColor.blackColor()
3      audioBarNum = 15;
4      for i in 0...audioBarNum {
5          let h:CGFloat = 150
6          let w:CGFloat = (self.view.frame.size.width-10)/
                       CGFloat(audioBarNum)
7          let x:CGFloat = 20
8          let y:CGFloat = 50
9          let view:UIView = UIView(frame:
                              CGRect(x: w*CGFloat(i)+x, y: y,
                              width: w-x, height: h))
10          self.view.addSubview(view)
      }
   }
```

代码第 1 行初始化 11 组随机颜色。第 2 行将 self.view 的背景颜色设置为黑色。第 3 行设置当前动画音量的柱状图的个数。第 4 行采用 for 循环的形式依次添加 15 根柱状图。代码第 5 到第 8 行设置柱状图的 frame 位置坐标。第 9 行初始化单根柱状图。最后一行将 15 根柱状图依次添加到 self.view 图层上。

2．动画进行阶段

既然每根音量柱状图都要实现图 10.15 的动画效果，所以在创建每个单根柱状图的时候都可以为其添加一个 CAGradientLayer 实例图层。以上 for 循环中完整代码如下所示：

```
for i in 0...audioBarNum {
          let h:CGFloat = 150
          let w:CGFloat =
          (self.view.frame.size.width-10)/CGFloat(audioBarNum)
          let x:CGFloat = 20
          let y:CGFloat = 50
          let view:UIView = UIView(frame:
```

```
                                CGRect(x: w*CGFloat(i)+x, y: y,
                                width: w-x, height: h))
            self.view.addSubview(view)
1           gradientLayer = CAGradientLayer()
2           gradientLayer.frame = view.bounds;
3           gradientLayer.startPoint = CGPoint(x:0,y:0);
4           gradientLayer.endPoint = CGPoint(x:0,y:1);
5           view.layer.addSublayer(gradientLayer)
6           layerArray.addObject(gradientLayer)
        }
```

代码第 1 行实例化一个 CAGradientLayer 实例对象，第 2 行到第 4 行设置 gradientLayer 图层 frame 位置属性，以及动画的运动方向。startPoint 和 endPoint 的设置可以保证动画如图 10.15 的运动效果。第 5 行为每个单独的柱状图的 Layer 图层添加一个 gradientLayer 实例对象，最后一行将所有的 gradientLayer 实例对象保存在 layerArray 数组中。

完成这一步之后动画也是无法启动的。想让动画模拟音量的跳动，而音量每时每刻又是在不停变化的，所以这里采用定时器的方式，模拟音量的随机变化。每次定时时间到当前柱状音量图就会随机展示图 10.15 的动画效果。具体实现代码如下。

```
    for i in 0...audioBarNum {
    柱状图添加代码（省略）
        }
 1      Timer.scheduledTimer(timeInterval: 0.4,
            target: self,
            selector: #selector(ViewController.colorChange),
            userInfo: nil,
            repeats: true)
    func colorChange(){
    2       for layer in layerArray{
    3          let index:Int = Int(arc4random_uniform(11))
    4           let color:UIColor = colorArray.object(at:index) as!
UIColor
```

```
5           let colors = [
              UIColor.clear.cgColor,
              color.cgColor
            ]
6           let layer = layer as! CAGradientLayer
7           layer.colors = colors
8           layer.locations = [0,1.0]
9           let gradientAnimation:CABasicAnimation =
                                              CABasicAnimation()
10          gradientAnimation.keyPath = "locations"
11          let beginValue = Float(arc4random_uniform(11))/10.0
12          gradientAnimation.fromValue = [beginValue,beginValue]
13          gradientAnimation.toValue = [1.0,1.0]
14          gradientAnimation.duration = 0.4
15          layer.add(gradientAnimation, forKey: nil)
        }
    }
```

该动画模拟音量每 0.4s 变化一次，所以代码第 1 行设置一个每 0.4s 执行一次的定时器，定时时间到则执行一次模拟音量变化动画效果。音量变化动画效果在方法 colorChange()中实现。代码第 2 行遍历我们为每根柱状图添加的 gradientLayer 图层。第 3 行和第 4 行从已经准备好的随机颜色数组中选取一组随机颜色。第 5 行到第 8 行设计 gradientLayer 图层颜色渐变效果中颜色和位置属性。第 9 行定义一个 CABasicAnimation 动画实例对象。第 10 行设置动画是对哪种动画属性起作用。这里是对颜色梯度变换中的 locations 属性起作用。第 11 行到第 13 行设置 locations 属性的变化范围。代码最后两行设置动画变化周期并将动画添加到 Layer 图层。

10.5 本章小结

CAGradientLayer 是 QuartzCore 框架下非常重要的一个类，该类在颜色的梯度变换中使用非常广泛。比如在本章实现的指纹光波扫描效果和音量跳动效果，

都是通过该类实现的。CAGradientLayer 掌握起来不是很容易，尤其是对光波的执行方向、颜色梯度、“彗星拖尾”效果的理解，但本章对其做了详细的原理和效果分析。大家可以结合第 10.2 节的知识打造一个个性化的颜色梯度效果，比如 iPhone 手机解锁中最常见的广播扫描解锁效果，如图 10.16 所示。

图 10.16　手机解锁光波动画效果