iOS UI绘制原理

高质量的图形展示在app的交互界面中扮演非常重要的角色。高质量的图形展示让用户更能喜欢使用它。iOS系统主要提供两种途径去创建高质量的图形：OpenGL或者使用原生Quarts、Core Animation和UIKit。本文会展开讲一下后者。

Quartz是主要的绘制途径，它提供了基于路径绘制、抗锯齿绘制、渐变色、图形绘制、颜色、变形和PDF文档的创建展示和解析能力。UIKit是对Quartz的线条、图片和颜色操作的封装。Core Animation提供了对在动画中修改UIView属性的的支持，同时还可以实现自定义动画。

这个章节会讲述iOS App中的渲染过程，并说明其中用到的绘制原理。可以帮你学习到一些可以让你自己app优化渲染的小技巧。

> 重要：并非所有UIKit的类都不是线程安全的。请确认在执行绘制的时候处于主线程。

UIKit 图形系统

在iOS中，无论使用哪种技术（OpenGL、Quartz、UIKit或者Core Animation）绘制在UIView或者子类中，都会在屏幕上展示。视图定义自己在屏幕上如何绘制，或者说如何展现自己。系统提供的视图会自动定义自己的展现。自定义视图你必须定义视图如何展现。本章就会讲解通过Quartz、Core Animation和UIKit去绘制自定义图形。

另外，可以离屏绘制位图和PDF图形上下文（译者吐槽：是不是意味着这部分可以不在主线程绘制？）。当你离屏绘制时，因为没有在视图上进行绘制，所以视图的生命周期也对此不起作用。

视图生命周期

UIView的子类包含了最基本的图形绘制模型 -- 根据需要更新绘制自身。UIView类通过批量更新绘制以及在合适的时机更新绘制自身来做到让更新自身绘制变得简单和高效。

当一个视图第一次或者某部分需要更新的时候iOS系统总是会去请求`drawRect:`方

法。

以下是触发视图更新的一些操作：

- 移动或删除视图
- 通过将视图的`hidden`属性设置为`NO`
- 滚动消失的视图再次需要出现在屏幕上
- 视图显式调用`setNeedsDisplay`或`setNeedsDisplayInRect:`方法

视图系统都会自动触发重新绘制。对于自定义视图，就必须重写`drawRect:`方法去执行所有绘制。在方法中通过原生绘制API来绘制本身的形状、文字、图片、渐变或者任何你希望展示的部分。视图第一次展示的时候，iOS系统会传递正方形区域来表示这个视图绘制的区域，**为了最大程度优化性能，重绘的时候最好只重绘受影响的部分**。

在调用`drawRect:`方法之后，视图就会把自己标记为已更新，然后等待下一次视图更新被触发。静态自定义视图需要处理因为滚动而出现或者因为其他视图出现而引起的视图变化。（译者吐槽：后半句啥意思？没太懂）

如果想要改变视图的内容，就必须触发视图重绘内容。通过调用`setNeedsDisplay`或者`setNeedsDisplayInRect:`方法来触发更新。使用场景例如一秒内多次更新视图，或者根据用户的交互行为在视图中出现新的内容。

> 重要：不要显式调用`drawRect:`方法。这个方法应该只留给iOS需要重新绘制的时候留给系统调用。因为在其他时机图形上下文是不存在的，所以也不能对屏幕进行绘制。（图形上下文下一个小节会说明。）

坐标系统及iOS中的绘制

当App需要在iOS系统中绘制图形时，它必须绘制在一个二维坐标系中。这看上去很简单，但在某些绘制情况下需要处理另一种不同的坐标系。

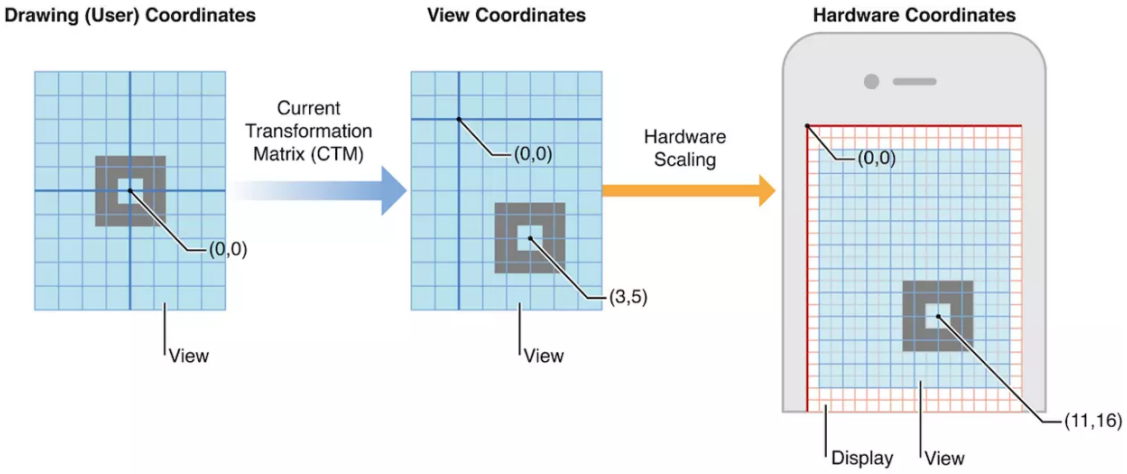
iOS的图形绘制需要依靠图形上下文来完成。理论上说，图形上下文是用来描述在哪里、如何画上，包括颜色绘制、切割绘制区域、线条粗细和样式等信息。

另外，图1-1中展示了每个图形上下文都有一个坐标系。更准确的说，每个图形上下都三个坐标系：

- 绘制坐标系。绘图上下文通过使用指令绘制的坐标。
- 视图坐标系。相对于视图的固定坐标系。

- 设备坐标系。物理屏幕的像素展示坐标。

图1-1 绘制坐标、视图坐标以及硬件坐标关系



译者注：CTM是Quartz中的一个概念，下面会介绍。

iOS的绘图框架为绘制特定的目标(屏幕、位图、PDF内容等)创建图形上下文，这些图形上下文为该目的地建立初始绘图坐标系。这个初始的坐标系被称为默认坐标系，是1比1映射到视图坐标系上的。（译者吐槽：后半句没懂）

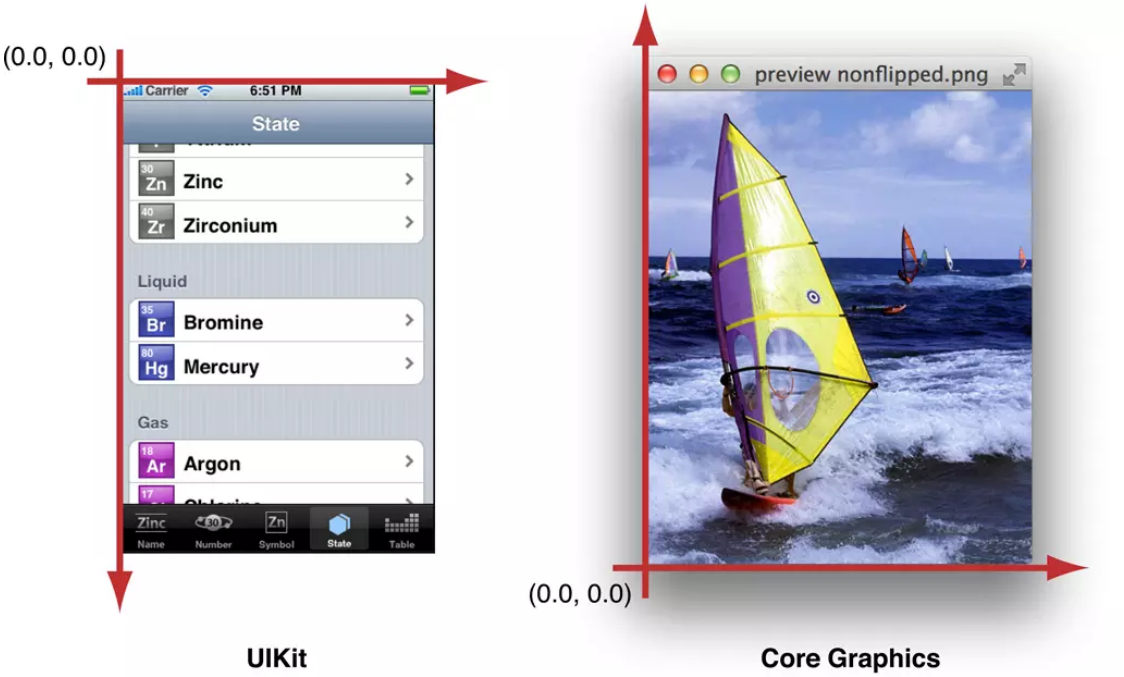
每个视图都有一个自己的`current transformation matrix(CTM)`，一个数字举证映射当前绘制坐标系到视图坐标系。App可以修改矩阵来影响后面发生的绘制操作。

iOS会在默认坐标系的基础上创建图形上下文。在iOS中主要是两种：

- 左上原点坐标系（ULO），从左上角为0,0坐标，向右向下为正，UIKit和Core Animation都是基于ULO。
- 左下原点坐标系（LLO），从左下角为0,0坐标，向右向上为正，Core Graphics是基于LLO。

两种坐标系展示如图1-2

图1-2iOS中默认坐标系



> 提示：MacOS默认使用的是LLO。通过AppKit和CoreGraphics绘制都是基于此坐标系，AppKit提供了左上原点坐标系的转换支持。

点和像素

iOS系统中指定的坐标系和底层设备绘制像素的之间有区别。当使用原生绘制列如Quartz，UIKit和Core Animation，绘制坐标西和试图坐标系都是逻辑坐标系（译者注：这里说的逻辑坐标系也就是指的不与设备像素点对应），坐标系数值表示的是**点**，与设备上的像素并没有一一对应的关系。

系统会自动根据视图的点坐标值去映射到设备的像素上，但并不一定是一对一映射，这点非常重要。

一个点不一定映射到物理的一个像素。

使用点代替的像素的主要目的还是为了让视图在设备上呈现出合适的尺寸，不会因为屏幕像素变高导致原本视图变得很小。具体多少像素对应一个点，是由系统根据当前设备硬件来决定的。例如，在视网膜屏幕上，一条线的绘制对应像个像素的线条宽度。这种映射关系让普通屏视网膜屏和更高分辨率的屏上展示视图的大小基本保持一致。

> 提示：Core Graphics中渲染和打印PDF的时候一个点点对应1/72英寸。

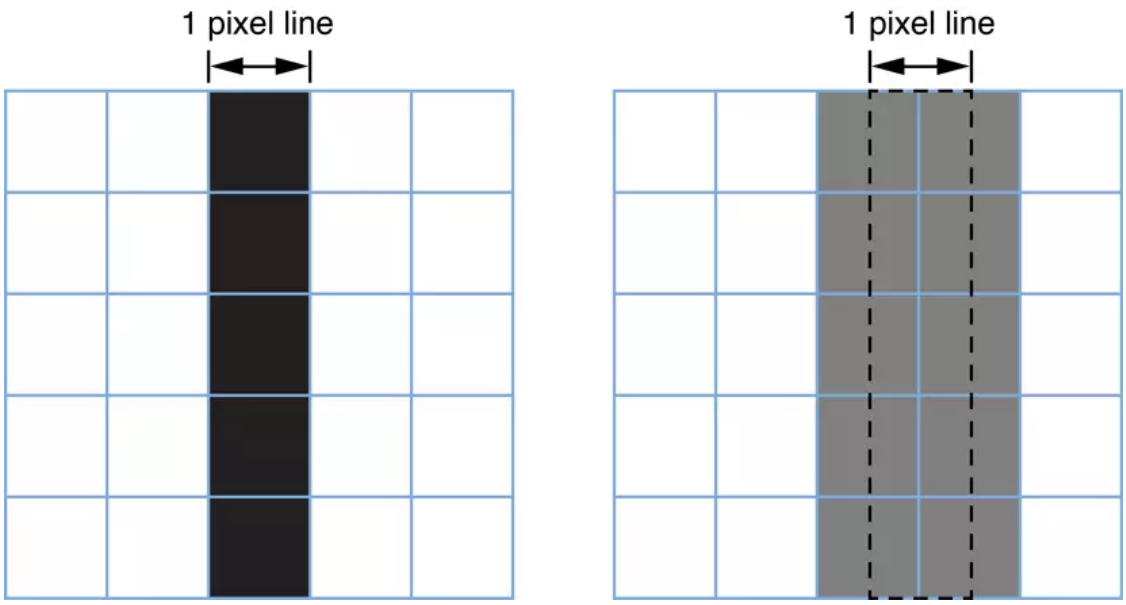
在iOS中，UIScreen，UIView，UIImage和CALayer都提供属性用于描述像素和点之

间的映射比例。例如，UIKit的View的`contentScaleFactor`属性。在非视网膜屏中，该属性值为1.0。在视网膜屏中，为2.0（译者注：除了plus系列后应该还有3.0）。在未来也可能出现其他的值。（在iOS4之前一直都是1.0）。

因为自动映射的关系，在绘制视图时通畅不需要关心像素。只有在下载高分辨率图片在视网膜屏幕上展示的时候，需要关心图片渲染的scale，避免高分图被低分渲染而变大的问题。

在iOS中，当你在屏幕上绘制东西时，图形子系统使用一种叫做反锯齿的技术，在低分辨率的屏幕上近似一个高分辨率的图像。用一个例子来解释下。绘制一条黑色的竖线在白色背景上，如果线正好落在像素上，展现出来就如下图左边那样是一系列黑色像素排列。如果正好落在两个像素上，那就会出现灰色像素绘制两格如下图1-3右侧。

图1-3



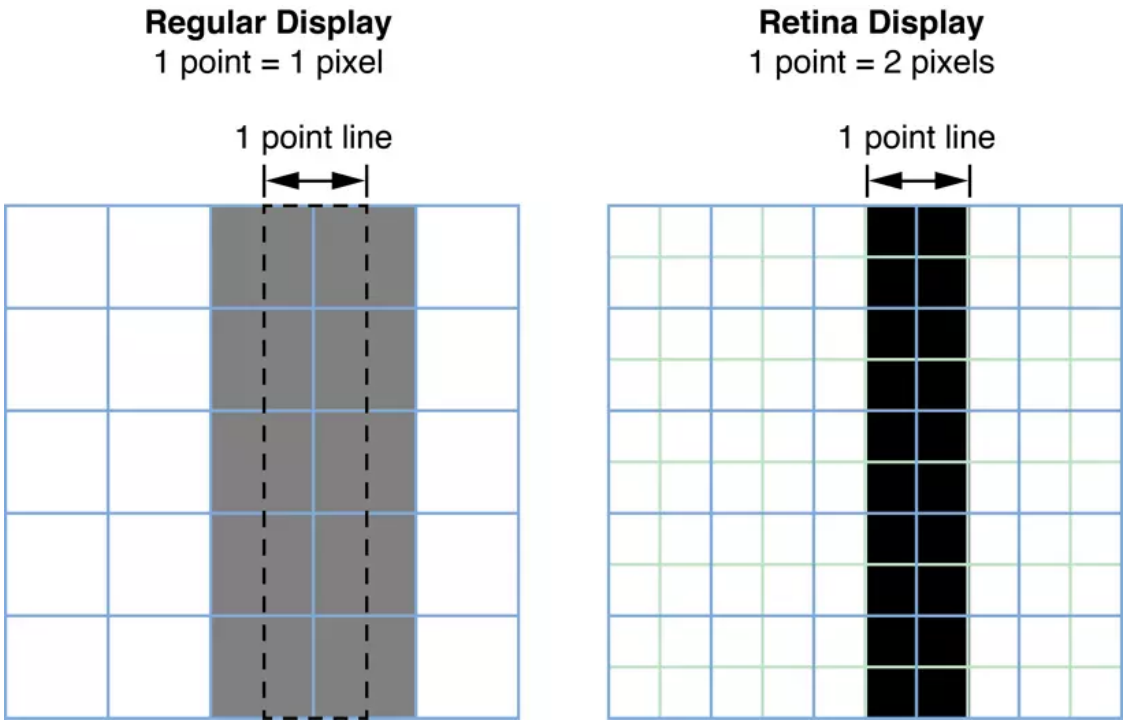
整数值的点坐标会落在两个像素的中间。例如，画一条1个像素宽度的直线(1,1)到(1,10)，得到的是一条灰色的先。如果画两个像素宽度的线，才会得到一条黑色的实线，因为两个像素正好落满两个像素。一般来说，如果不调整它们的位置，使它们完全覆盖像素，那么与宽度为偶数的物理像素的宽度相比，奇数个物理像素宽的线显得更浅。

scale属性就是为了表示一个点映射了多少像素。

在非视网膜屏上scale永远为1.0，一个点对应一个像素。为了避免反锯齿，当你绘制一个单点线时，如果占了奇数整数宽度，那就需要偏移0.5个点，如果占用偶数宽

度则不必这么做。

图1-4 一个点宽度的线在非视网膜和视网膜屏上的展示



在视网膜的scale为2.0，一点线也不会触发抗锯齿，因为本身就会撑满两个像素。如果要画一条一个像素的线，就需要使用0.5个点的宽度并且偏移0.25个点。

直接按照scale去控制像素绘制并不能得到最好的体验。一个一像素宽的线在非视网膜屏幕上看起来可能没问题，但如果在视网膜屏幕上看起来就会觉得太细了。这取决于你如何去绘制。

获取图像上下文

图像上下文可以在`drawRect:`方法中获取到，并且立刻进行绘制。UIView为图像上下文提供了绘制的环境。

如果您想在视图以外的地方绘制(例如，在一个PDF或位图文件中捕获一系列绘图操作)，或者如果您需要调用需要上下文对象的核心图形函数，那么您必须采取额外的步骤来获取图形上下文对象。下面的章节解释了为什么。

更多关于修改图像上下文状态和创建定制内容请参考 [Quartz 2D Programming Guide]。图像上下文的方法清单请参考 [CGContext Reference], [CGBitmapContext Reference], [CGPDFContext Reference]

在屏幕上绘制

想要在屏幕上绘制，就需要在`drawRect:`方法中获取到图像上下文。（这一系列方法中的第一个参数都是一个`CGContextRef`对象。）可以通过调用`UIGraphicsGetCurrtnContext`方法，在`drawRect:`方法中获取一个图形上下文。（多次获取也会得到同一个。）

在UIKit的view中，使用Core Graphics系列方法来绘制是基于ULO坐标系的。或者，翻转CTM来使用LLO坐标系来绘制。详细的请阅读 [Flipping the Default Coordinate System]

UIGraphicsGetCurrentContext函数始终返回的是当前的上下文。例如，在创建PDF后获取的上下文就是PDF上下文。只要是使用Core Graphics系列函数绘制，都必须使用这个方法来获取上下文。

> **提示：** 打印相关的函数放在了`UIPrintPageRender`类中。类似于`drawRect:`，UIKit在其中提供了打印相关的实现。并且默认也是基于ULO坐标系。

绘制位图和PDF

UIKit提供了绘制位图和PDF的上下文和系列函数。两种创建方式都需要分别调用一个函数来创建其对应的上下文。在通过上下文进行绘制，并在绘制完成后关闭上下文。

两种上下文也是基于ULO坐标系。Core Graphics提供了一系列方法用于在在位图上下文中徐然和在PDF上下文中绘制。上下文从Core Graphics中直接调用函数获得，并且基于LLO坐标系绘制。

> **提示：**在iOS中还是推荐使用UIKit的相关函数来获取上下文来绘制。如果非要使用Core Graphics中的相关方法来绘制，则需要对坐标系的差异做兼容。（译者：所以UIKit中的上下文相关方法其实是转换了坐标系的Core Graphics方法。）

详细可以参考 创建绘制位图 和 创建PDF

颜色和色域

尽管Quartz在iOS系统中支持全色域；但是几乎所有app中都只用到了RGB色域。毕竟iOS被设计在屏幕上绘制渲染，而RGB是最合适的。

UIColor对象通过提供一系列便捷方法通过RGB/HSB和灰度色值来创建颜色，且不需要关心色域问题，而由UIColor对象自动决定。

也可以使用Core Graphics框架中的`CGContextSetRGBStrokeColor`和`CGContextSetRGBFillColor`函数来创建社设置颜色。尽管Core Graphics提供了可以指定色域和创建自定义色域的函数，但是并不推荐在代码中使用。（译者：为啥不推荐？原因呢？）还是推荐始终使用RGB色域即可。

使用Quartz和UIKit绘制

我们把iOS中的绘图技术统称为Quartz。而Core Graphics框架则是Quartz心脏，并承担大多数绘制内容的职责。框架提供了数据类型和函数支持以下能力：

- 图形上下文
- 路径
- 图片和位图
- 透明图层
- 颜色和色域
- 渐变和阴影
- 字体
- PDF

UIKit在Quartz基础上提供了一套图形操作相关的类。目的并不是为了替代Core Graphics，相反，他们是为了给UIKit的其他类提供绘画支持：

- UIImage/UIColor/UIFont/UIScreen/UIBezierPath
- 生成一个JPEG或PNG的图片对象的函数
- 获取位图上下文的函数
- 获取PDF上下文的函数
- 绘制矩形和裁剪绘图区域的函数
- 获取当前图形上下文的函数

更多信息参考，UIKit Framework Reference，还有 Core Graphics Reference。

配置图形上下文

在调用`drawRect:`方法之前，视图对象已经创建了一个图形上下文并且将其置为

当前的上下文。它只存在于`drawRect:`方法调用期间。可以通过调用`UIGraphicsGetCurrentContext`函数来获取图形上下文的一个引用。方法返回一个`CGContextRef`类型对象的引用，该对象传递了Core Graphics函数修改当前图形的状态。表1-1列出了主要的几个方法。需要查看完整的请移步 CGContext Reference。下表还列出了UIKit替代方法。

表1-1 修改图形状态的Core Graphics方法

状态	函数名	UIKit替代方法
Current transformation matrix (CTM)	CGContextRotateCTM/CGContextScaleCTM/CGContextTranslateCTM/CGContextConcatCTM	None
Clipping area	CGContextClipToRect	UIRectClip function
Line: Width, join, cap, dash, miter limit	CGContextSetLineWidth/CGContextSetLineJoin/CGContextSetLineCap/CGContextSetLineDash/CGContextSetMiterLimit	None
Accuracy of curve estimation	CGContextSetFlatness	None
Anti-aliasing setting	CGContextSetAllowsAntialiasing	None
Color: Fill and stroke settings	CGContextSetRGBFillColor/CGContextSetRGBStrokeColor	UIColor class
Alpha global value (transparency)	CGContextSetAlpha	None
Rendering intent	CGContextSetRenderingIntent	None
Color space: Fill and stroke settings	CGContextSetFillColorSpace/CGContextSetStrokeColorSpace	UIColor class

Text: Font, font size, character spacing, text drawing mode	CGContextSetFont/CGContextSetFontSize/CGContextSetCharacterSpacing	UIFont class
Blend mode	CGContextSetBlendMode	The UIImage class and various drawing functions let you specify which blend mode to use.

上下文中以堆栈形式保存了图形的装填。上下文被Quartz创建时堆栈是空的。通过调用`CGContextSaveGState`函数将当前图形状态推入堆栈。此后图形状态的改变会影响后续的绘制操作，但不会影响之前已如堆栈的。当完成修改后可以通过调用`CGContextRestoreGState`函数来将其中堆栈中弹出。这种推入和弹出操作替代了逐个撤销每个状态的操作。这也是唯一能还原到之前状态的方法。

更多信息参考 Graphics Context 和 Quartz 2D

绘制路径

路径是一系列线和贝塞尔曲线组成的矢量形状。UIKit中包含了`UIRectFrame`和`UIRectFill`等函数用于绘制简单的路径（类似矩形）。Core Graphics也提供了便捷函数用于绘制简单路径（例如矩形和椭圆）。

更多复杂路径，就需要使用`UIBezierPath`类自己画了，或者使用函数操作Core Graphics提供的`CGPathRef`。尽管可以脱离上下文绘制路径，但最终底层还是使用了上下文，只是被封装了。

--- 结束 ---