## 简介

Quartz 2D是2D绘图引擎，适用iOS和Mac OS X程序。它提供底层轻量级的接口，并且会根据输出显示设备提供不匹配的逼真的画面 。Quartz 2D是与分辨率和设备无关的一个引擎。你不需要关注最终成像目标位置如何。非常容易使用，提供大量强大的功能，例如透明度，画线，视角距离效果渲染，颜色管理，反锯齿渲染，PDF文档创建，显示，解析。Quartz 2D是Core Graphics框架的一部分，所以看到CG开头，别奇怪，就是Core Graphic的简写。

## Quartz 2D 概览

在MAC OS系统中，QUARTZ 2D能够与其他一些图形处理技术混用，例如Core Image, Core Video, Open GL , QuickTime，你可以从QUICKTIME视频中创建图片，你可以把数据提供给Core Image来进行图形处理。

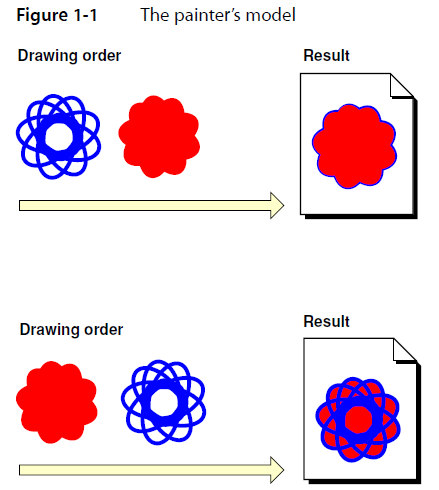
在IOS里，QUARTZ 2D跟其他很多动画技术混用，例如Core Animation, Open GL ES, Uikit。

哥忍不住插一句，Core Animation以及Quartz乃苹果完美的UI体验的后台支持啊。邪恶的苹果，把Pixal动画公司的电影级动画制作技术，带到了苹果IOS里。比如Uinavigation，pushViewController，无不是animation在后台起作用，那些左右滑的体验，有木有想过如何一句话就能实现？其实这个呈现技术，跟Flash还真就差不多，所以两家公司暗战难免，核心技术一样啊！

## The Page 介个你叫我如何翻译是好？？页？

Quartz 2D使用了一个常人绘画者的模型，每一次成功的操作绘图，都是在之前的基础上层叠。绘上去的，不能再改。除非再绘图改掉那一块。无法撤销，永远是增量式改进。

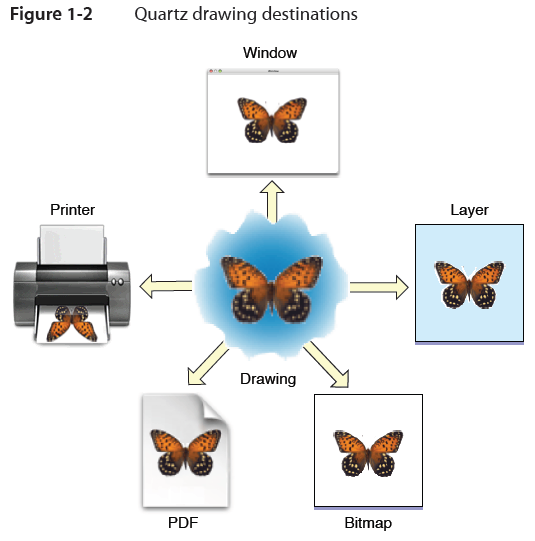
好吧，你们看下图，就懂了，绘图顺序不同，导致了不同的结果。



这个所谓的Page页，不一定是一张纸，可能是打印机，可能虚拟的pdf纸张概念，可能是bitmap，到底是啥取决于你使用哪一种context。记住，以后可能N种类型的Context，比如CGPDFContext，CGBlahblah…Context等等。

## 绘画到哪儿去了？ Graphics Context。

一个graphics context是一个不透明的图形数据类型，我们有个基类叫CGContextRef，它概括了这个图形所有的内容，可能是一个pdf文件，一个bitmap，一个显示着的窗口。它内部包含了这么些信息，绘图参数，设备指定的显示参数比例（比如pdf）。你不需要考虑设备投放比例等等问题，quartz替你做好了。很高档。



这些graphics contexts有如下特殊的分支类型：

● 一个bitmap的graphics contexts允许你绘RGB颜色，CMYK颜色，bitmap里的灰度值。Bitmap嘛，大家都懂的，微软搞的格式，苹果也差不多一个意思，总结下来，就是每个像素点，一个颜色，最终汇集为一个Array集合。我不禁想起，大学毕业设计搞的游程图像编码，唉。

● 一个PDF类型的graphics context嘛，允许你创建PDF文档，绘出来。它与BITMAP类型的明显区别就是：

● PDF 文件可能存在很多页，介个。。我就不解释了。

● 当你绘图一个pdf时候，它会根据你最终输出的设备像素大小，来动态优化，也就是常说的，显示百分比例。

● PDF文件是天生的分辨率独立的东西，就是说，矢量图啦。可以放大，缩小，而不损失分辨率。我想起了UI的PSD，差不多一个意思吧。而BITMAP呢，由显示多大决定了自身的清晰度，会经过一系列处理损失自身分辨率。

● 一个window 类型的graphics context，就是能绘画到一个窗口里的类型啦。注意，这个Quartz 2D图形引擎不是窗口管理系统，想要从一个窗口获得graphic context，请参考其他文档， See “Creating a Window Graphics Context in Mac OS X” (page 29) for details.

● 一个layer类型的context，类名叫CGLayerRef，差不多意思是，是一个层面的概念，一个context里，可以有很多CGLayer层，一层一层，重叠起来，注意，不透明。

## Quartz 2D 各种不透明的数据类型介绍

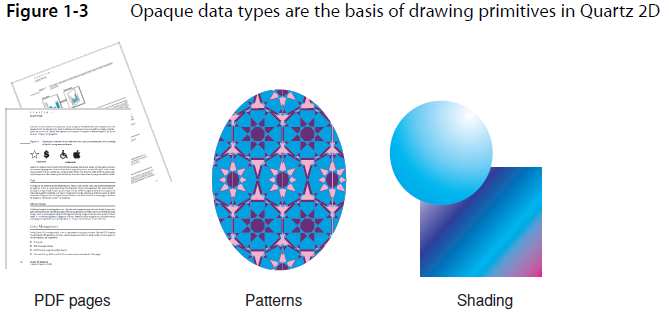
Quartz 2D API定义了相当多的数据类型，这些都是Core Graphic的框架下，所以都是CG开头命名。

你想要实现某功能，请找相应的类去实现。下面举例：

● 你可以利用PDF PAGE对象（也就是CGPDFContext），旋转显示PDF的一页，并且让Quartz 2D去渲染显示到graphics context里。

● 你可以画一个样板，创造一个自定义的对象，定义模样，并且要求按照指定的模样画进去。

● 你也可以在一个轴状，或者放射状的区域里填充颜色，提供一个方法去依据每个点判断决定颜色，最后填充满。



以下是具体类名介绍:

● CGPathRef, 使用矢量图形来画线。

● CGImageRef, 用来呈现bitmap或者bitmap的面具mask，面具的渊源在于bitmap时代，没有不透明的概念。所以，如果想层叠怎么办？就出现了bitmap与bitmap mask，简单说，就是黑透明，白不透明。

● CGLayerRef, 用来呈现层面，可能会被重复多次使用。例如背景墙，patterns等等。

● CGPatternRef, 用来重复绘画。哥哭了，尼玛这个与CGLayerRef有嘛区别啊？

● CGShadingRef and CGGradientRef, 用来绘画曲线。

● CGFunctionRef, 用来定义指定了曲率作为参数的回调函数。当你创造一个曲线对象的时候，拿这个CGFunctionRef 作为函数指针传递过去。

● CGColorRef and CGColorSpaceRef, 用来告诉Quartz引擎如何去理解颜色。简要说，就是颜色啦。

● CGImageSourceRef and CGImageDestinationRef, 这个很重要哦，是Quartz通向外界的进出口，所有的进出数据类型，都指望它呢。

● CGFontRef, 画TEXT. 不禁想UIlabel是不是就是封装了这个而已。

● CGPDFDictionaryRef, CGPDFObjectRef, CGPDFPageRef, CGPDFStream, CGPDFStringRef, and

CGPDFArrayRef, 提供了PDF所有的数据访问操作。

● CGPDFScannerRef and CGPDFContentStreamRef, 提供了解析PDF的功能。

● CGPSConverterRef, 用来PostScript转化为PDF，注意IOS中不存在。

## Graphics States

## 图形状态

Quartz根据图形声明的参数来修改绘画动作的结果。比较生涩，简单举例来说吧：

当你用一个功能，填充满颜色，你就是在修改一个图的声明。这么说吧，photoshop玩过吧，每次你修改一次，都生成一个新的图片层。

以此类推，quartz也有这么一个图形栈的说法，stack。妈呀，我想起了symbian的恶心的push,pop栈。

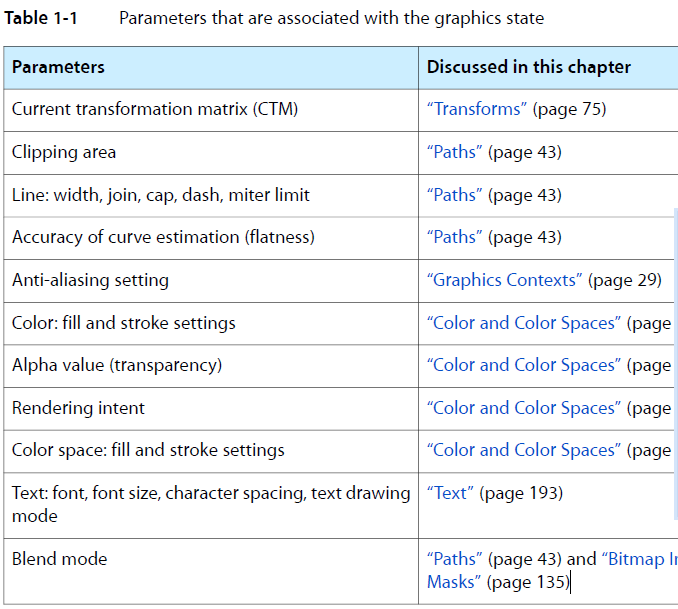
图形当你创建的时候，它状态是empty空，当你保存了一次，他就把你的状态生成一个备份到声明列表里。

保存当前图形状态，使用CGContextSaveGState来push进去一个状态。

当你想恢复原先的图形状态，就需要pop出来了，使用CGContextRestoreGState来实现。Pop出来的，变为当前默认最新状态。

注意，不是绘图环境里的每个方面的绘图，会作为图形声明。例如，path不会作为图形声明，当你使用CGContextSaveGState保存的时候，path不会保存起来。

你所有需要保存声明的参数，需要见如下表，来对应使用。



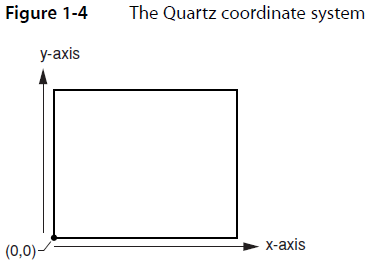
## Quartz 2D坐标系

A coordinate system, shown in Figure 1-4, defines the range of locations used to express the location and

sizes of objects to be drawn on the page. You specify the location and size of graphics in the user-space

coordinate system, or, more simply, the user space. Coordinates are defined as floating-point values.

见图，定义了坐标的范围和尺码。它的空间，不是指屏幕的真实坐标系，是存在用户空间范围内的坐标系。所以，会存在嵌套空间坐标。数值都是float类型。



因为不同的设备有着不同的成像能力，所以坐标系必须与设备无关。可能会存在每寸96个像素点，也可能存在每寸300个像素点。所以，你如果只是依照某一个设备来绘图，那么你可能无法挪到其他设备上跑。

譬如iphone，ipad，iPhone4 Retina显示屏。不得不说，这些底层的API定义的考虑非常广，也不一定指iphone，ipad，定义这些API的时候，能够想到未来可能会存在IPAD。真实棒。

Quartz完成不同设备的转换。它使用“用户空间”的概念。它通过一个“转换模型transformation matrix”或者CTM来完成这个功能。Matrix即模型，是一个数学搭建的模型，能够快速执行转换的方程式。当前用的“转换模型transformation matrix”是一个特殊的模型，简称“投影变换”。它能从一个坐标空间，映射到另外一个空间，可以翻译过去，可以旋转扭曲过去，放大缩小过去。牛逼吧。

这个投影变换，还有第二个用途：能够转化掉一个对象本来的绘出来的方式。

例如，画个盒子旋转45度，你在画盒子之前就旋转坐标系45度。Quartz就会按照你的坐标系投射到设备上去，效果就达到了。

一个点在坐标系里，简称（x,y），X指水平线坐标，Y指竖线坐标。（0,0）则是在左下角。

这个似乎跟UIWindows不同，那个是在左上角。

有些技术设置了不同的坐标原点。比如cocos2d。当你想让两个坐标系混用的时候，你必须对Quartz作出补偿，采用Quartz世界里的坐标。

有这么些场景需要变换，你可能已经碰到过了：

● In Mac OS X, a subclass of NSView that overrides its isFlipped method to return YES.

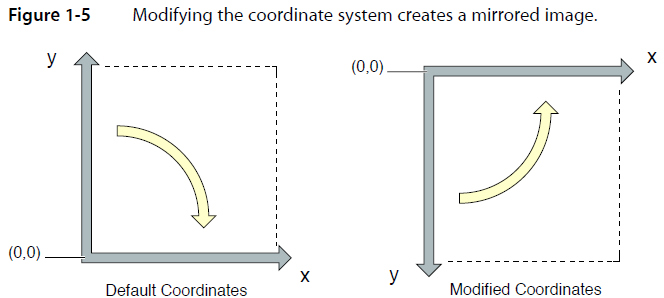
● In iOS, 一个UIview返回的context

● In iOS, 调用UIGraphicsBeginImageContextWithOptions来创造的绘图context对象。

其实还有好多好多，坐标系要互相妥协嘛。比如你在cocos2d中弹个UIVIEW，同样需要作出妥协补偿。这个原理是相通的。

如果你想同时展示一个pdf context和UIView，那么你需要使用“投影变换”了。说白了，就是scale一下，Y坐标通通乘以-1。

有些废话，我就不翻译了。突然觉得国语好省。



所以调整不调整，还是看你自己。

如果你希望一个image或者pdf直接绘图到显示设备，你的程序需要临时的调整一下CTM。在iOS里如果你用UIImage里面包着CGImage，你不需要调整CTM。UIImage会自动转换。

重要：上面的讨论，都是仅在你想直接写到iOS上去，但是并不是必要条件。在iOS 3.2之后，UIKit创造context的时候，会自动转化。不过pattern , shadows是例外，你需要手动调整。

## 内存管理：对象的关系

引用计数reference cout还是一样沿用过来了。我就不写了。

几个原则记得就行：

● 谁创造，谁释放。

● 你如果调用第三方API，无关创造，复制，那您别多事儿，没事儿别release。

● 如果你不拥有它，却要留着它在身边用，你必须retain和release一下。 例如你创造了一个CGColorspace对象，你使用了CGColorSpaceRetain和对应的CGColorSpaceRelease。你也可以使用CFRetain 和 CFRelease, 不过你得小心别传NULL到这些函数里去。

## Graphics Contexts

图形

Graphics Contexts代表一个绘图目的地。其包含绘图参数和所有与设备相关的信息，这些在绘制系统执行绘制命令的时候是需要的。Graphics Contexts定义了基本的绘制属性，如绘制的颜色，clipping 区域，线的宽度，样式信息，字体信息，compositing options,以及其他的。

我们可以使用Quartz context创建函数或者IOS的UIKit框架去得到一个Graphics Context。Quartz提供了几种风格的Quartz Graphics Contexts，包括bitmap和PDF。

下面本文主要讲述如何为各种各样的绘制目标创建Graphics Contexts。Graphics Context在代码的中的数据类型为CGContextRef，是一个不透明的data type。当得到一个Graphics Context之后，可以使用Quratz 2D函数去绘制context，在context上进行一些操作（如translations），改变graphics 的状态参数，如line的宽度和fill color。

## IOS中在一个view Graphics Context上绘制。

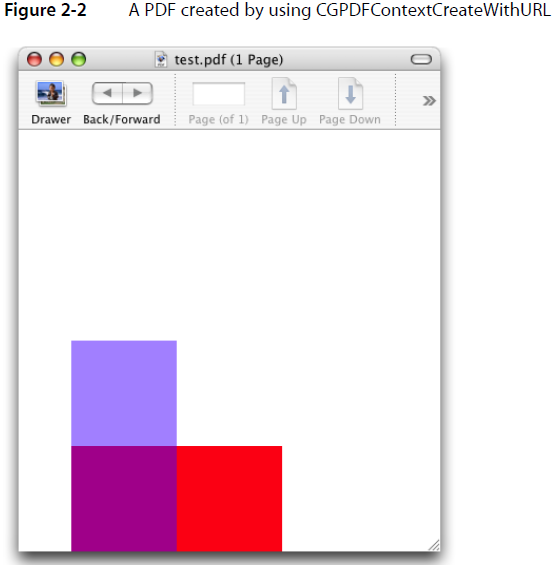
为了在一个ios应用上面绘制，应该先建立一个UIView对象，然后实现其drawRect:方法去进行绘制。当view变的可见的时候或者其内容需要更新的时候，view的drawRect:方法会被调用。在调用自定义的drawRect：方法之前，view对象会自动的配置其绘制环境，然后代码可以立即的绘制。作为配置的一部分，UIView对象先为当前的绘制环境创建一个Graphics Context。我们可以在drawRect:方法中通过调用UIKit的UIGraphicsGetCurrentContext.函数去得到此Graphics Context。

UIKit使用的缺省坐标系统与Quartz的坐标系统不相同。在UIKit中，原点在左上角。UIView对象通过translating 原点并且通过在y轴乘以-1来改变Quartz Graphics Context使其与UIKit相匹配。

## 创造一个 PDF Graphics Context

你给一个URL，它就能创造一个pdf graphic context。

如下图：



提供两种方法来创建对象：

● CGPDFContextCreateWithURL, 给个pdf的URL地址即可。网络OR本机，均可。

● CGPDFContextCreate, 直接创建一个空的pdf对象。

Listing 2-2 使用 CGPDFContextCreateWithURL 创建 PDF graphics context

CGContextRef MyPDFContextCreate (const CGRect \*inMediaBox,

CFStringRef path)

{

CGContextRef myOutContext = NULL;

CFURLRef url;

url = CFURLCreateWithFileSystemPath (NULL, // 1 path, kCFURLPOSIXPathStyle,

false);

if (url != NULL) {

myOutContext = CGPDFContextCreateWithURL (url, // 2

inMediaBox,

NULL);

CFRelease(url); // 3

}

return myOutContext; // 4

}

以上代码实现了如下功能:

1. 使用CFString来创造CFURL，然后使用NULL来作参数， 代表使用默认allocator分配，然后检查文件是否存在。

2. 生成PDF graphics , URL是地址，inMediaBox指CGRect为默认的。

3. 释放 CFURL object.

4. 返回PDF graphics context. 调用它的人，必须释放它。

## 这里介绍直接创建一个PDF文档

**Listing 2-3**  Calling CGPDFContextCreate to create a PDF graphics context

|  |
| --- |
| CGContextRef MyPDFContextCreate (const CGRect \*inMediaBox, |
| CFStringRef path) |
| { |
| CGContextRef myOutContext = NULL; |
| CFURLRef url; |
| CGDataConsumerRef dataConsumer; |
|  |
| url = CFURLCreateWithFileSystemPath (NULL,  // 1 |
| path, |
| kCFURLPOSIXPathStyle, |
| false); |
| //这里实现了创建文件的路径URL |
| if (url != NULL) |
| { |
| dataConsumer = CGDataConsumerCreateWithURL (url);  // 2  //2 这里实现了通过URL生成quartz data注意,不是quartz 文件 |
| if (dataConsumer != NULL) |
| { |
| myOutContext = CGPDFContextCreate (dataConsumer,  // 3  //3 生成了pdf context,注意它使用父类通用的定义的类型哦 |
| inMediaBox, |
| NULL); |
| CGDataConsumerRelease (dataConsumer);  // 4 |
| } |
| CFRelease(url);  // 5 |
| } |
| return myOutContext;  // 6  //返回pdf 的context |
| } |

## 这里介绍一下如何绘画一个PDF文件

**Listing 2-4**  Drawing to a PDF graphics context

|  |
| --- |
| CGRect mediaBox;  // 1 |
|  |
| mediaBox = CGRectMake (0, 0, myPageWidth, myPageHeight);  // 2  //所谓的mediaBox就是CGRECT俗称空间啦 |
| myPDFContext = MyPDFContextCreate (&mediaBox, CFSTR("test.pdf"));  // 3  //此处是上一个方法 |
|  |
| CFStringRef myKeys[1];  // 4  // 对了,PDF是书,存在页码的概念,所以必须有这么一个字符串数组来决定有多少页 |
| CFTypeRef myValues[1]; |
| myKeys[0] = kCGPDFContextMediaBox; |
| myValues[0] = (CFTypeRef) CFDataCreate(NULL,(const UInt8 \*)&mediaBox, sizeof (CGRect)); |
| CFDictionaryRef pageDictionary = CFDictionaryCreate(NULL, (const void \*\*) myKeys,  (const void \*\*) myValues, 1,  &kCFTypeDictionaryKeyCallBacks,  & kCFTypeDictionaryValueCallBacks); |
| CGPDFContextBeginPage(myPDFContext, &pageDictionary);  // 5 |
| // \*\*\*\*\*\*\*\*\*\* Your drawing code here \*\*\*\*\*\*\*\*\*\*  // 6  //好吧,那你开始绘图了. |
| CGContextSetRGBFillColor (myPDFContext, 1, 0, 0, 1); |
| CGContextFillRect (myPDFContext, CGRectMake (0, 0, 200, 100 )); |
| CGContextSetRGBFillColor (myPDFContext, 0, 0, 1, .5); |
| CGContextFillRect (myPDFContext, CGRectMake (0, 0, 100, 200 )); |
| CGPDFContextEndPage(myPDFContext);  // 7  //结束绘图 |
| CFRelease(pageDictionary);  // 8 |
| CFRelease(myValues[0]); |
| CGContextRelease(myPDFContext); |

以上你应该能学会如何制作一个PDF出来,也就是说,程序是有能力按照意图来生成PDF文档的.

## 生成BITMAP CONTEXT

Bitmap graphics context接受一个指向一定内存区域的指针来表达像素点阵.当你画的是好,数组会自动更新到最新的实时数据上去.

IOS注意:请使用UIGraphicsBeginImageContextWithOptions这个API,别用太底层的API,总归有什么坐标系不同,blah…blah…blah.反正你用这个就对了.好吧,下面很多是MAC OS开发的介绍,我就没兴趣了.

这个API,注意,声明就在UIkit.h里,所以,你懂的,下面摘抄部分给大家看看UIKIT的API列表

Image and Movie Saving

* UIImageWriteToSavedPhotosAlbum
* UISaveVideoAtPathToSavedPhotosAlbum
* UIVideoAtPathIsCompatibleWithSavedPhotosAlbum

Graphics

* UIGraphicsGetCurrentContext
* UIGraphicsPushContext
* UIGraphicsPopContext
* UIGraphicsBeginImageContext
* UIGraphicsBeginImageContextWithOptions
* UIGraphicsGetImageFromCurrentImageContext
* UIGraphicsEndImageContext
* UIRectClip
* UIRectFill
* UIRectFillUsingBlendMode
* UIRectFrame
* UIRectFrameUsingBlendMode

PDF Creation

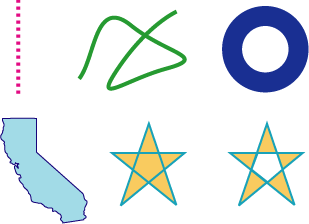
* UIGraphicsBeginPDFContextToData
* UIGraphicsBeginPDFContextToFile
* UIGraphicsEndPDFContext
* UIGraphicsBeginPDFPage
* UIGraphicsBeginPDFPageWithInfo
* UIGraphicsGetPDFContextBounds
* UIGraphicsAddPDFContextDestinationAtPoint
* UIGraphicsSetPDFContextDestinationForRect

UIGraphicsSetPDFContextURLForRect

## 画线

**Figure 3-1**  Quartz supports path-based drawing

Quartz 支持基于线条的绘画



## Path Creation and Path Painting

## 路线的创建和绘图

创建路线和绘图线,是两部分工作.首先你创建路线.你需要渲染的时候,请出来Quartz帮忙.

线路,各种各样,圆圈,方形,曲线,乱七八糟的线.

下面介绍几种线条的基本元素:

1. Points 点

系统通过[CGContextMoveToPoint](file:///Library/Developer/Documentation/DocSets/com.apple.adc.documentation.AppleLion.CoreReference.docset/Contents/Resources/Documents/documentation/GraphicsImaging/Reference/CGContext/Reference/reference.html" \l "//apple_ref/c/func/CGContextMoveToPoint) 来画线,系统永远实时追踪current Point, 当前最新的点.你可以MOVE到这个CGPoint,再MOVE到另外一个点.注意,线不是由点组成的.你画点,只是一个起点的概念.是用了画其他线条之类的起点概念.

1. Lines

所谓画线嘛,永远是从current point开始咯,你只需要制定最后结束点是哪儿就可以了.

请直接使用[CGContextAddLineToPoint](file:///Library/Developer/Documentation/DocSets/com.apple.adc.documentation.AppleLion.CoreReference.docset/Contents/Resources/Documents/documentation/GraphicsImaging/Reference/CGContext/Reference/reference.html" \l "//apple_ref/c/func/CGContextAddLineToPoint) 这个API,就搞定啦.

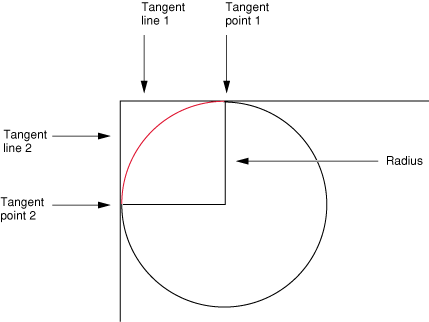
1. Arcs

好了,各位雷神,现在轮到画弧线,圆圈了.

用[CGContextAddArc](file:///Library/Developer/Documentation/DocSets/com.apple.adc.documentation.AppleLion.CoreReference.docset/Contents/Resources/Documents/documentation/GraphicsImaging/Reference/CGContext/Reference/reference.html" \l "//apple_ref/c/func/CGContextAddArc) 这个API,指定圆心,半径,还有神马…好像没有了.圆周率大家该不会认为系统也需要指定吧.嘿嘿

还有一个API, [CGContextAddArcToPoint](file:///Library/Developer/Documentation/DocSets/com.apple.adc.documentation.AppleLion.CoreReference.docset/Contents/Resources/Documents/documentation/GraphicsImaging/Reference/CGContext/Reference/reference.html#//apple_ref/c/func/CGContextAddArcToPoint) 这个呢, 参考下图,如果你不想 全画出来一个圈,你就需要指定半径,起点,终点.

注意哦,有木有发现这里没有需要指出来圆心? 呵呵,那当然咯,Quartz替我们干好了.你们该不会不知道在这个情况下如何算圆心坐标吧..嘿嘿



1. Curves 无规则曲线

这个就必须得用正规的方程式来算出来到底该画哪个点了.

总觉得这个越来越不着边际,往偏了写了.我就不继续翻译Path了.

## Transforms

## 变换技术

原理先解释下,每个user space的坐标系,都是与最终输出的设备分辨率无关的.QUARTZ能够替你做这部分工作. 我们把current transformation matrix叫CTM,当前的转化矩阵.

说白了,就是你当前的graphics context的容器.如果我们对CTM进行修改, BINGO,你就实现了扭曲,变换,各种神奇的效果就来了.

默认有5种变换方法, 旋转,转化,放大缩小,连接几个页面,好吧,我数了一下,其实只有3种.

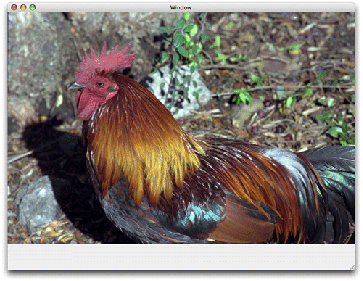
好吧,下面一个个介绍咯.

首先假定有这么个CGImage对象,我们看看在下面几种转化过程中,不同的API都做出了不同的效果:

CGContextDrawImage (myContext, rect, myImage);

**Figure 5-2**  An image that is not transformed

此图没经过任何转化处理

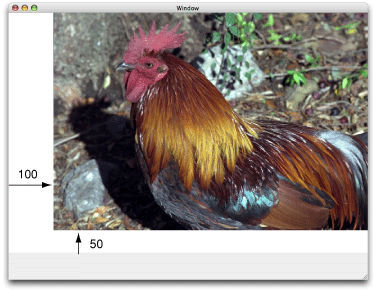


对它进行尺寸修改

CGContextTranslateCTM (myContext, 100, 50);

**Figure 5-3**  A translated image

往某方向移出范围.记住,这个类似剪切部分图片.



大家看好了哦,Translation的效果就是这个意思,注意对比原图,鸡尾巴没有了…啊,我差点打成鸡巴….~!@#$%^&\*()\_

下面给大家看Rotation的效果

CGContextRotateCTM (myContext, radians(–45.));

#include <math.h>

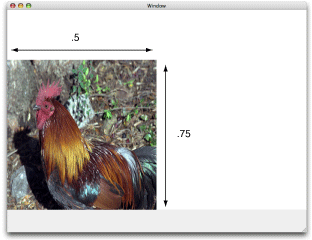
static inline double radians (double degrees) {return degrees \* M\_PI/180;}



基本效果就是酱紫,旋转了45度,部分out of这个context范围了.为嘛写这个内联函数呢?省事儿,下次调整个角度就方便多了.

好吧,下面介绍放大缩小.

CGContextScaleCTM (myContext, .5, .75);



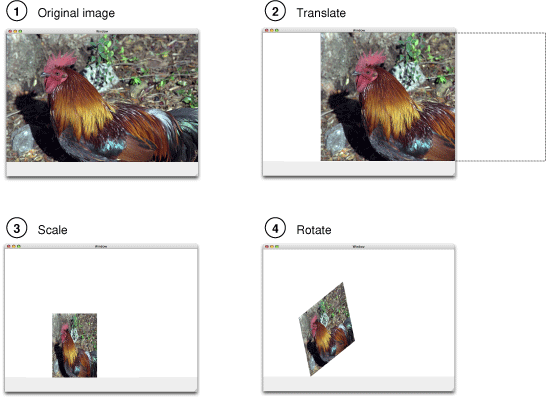
注意,两个参数,0.5是比例,0.75也是比例的意思.

Concatenation

我仔细理解之后,发现,这TMD就压根不是字面的组合的意思.说白了.这就是批量化执行脚本,能够省去你一大堆的活儿. 的确,很多图,先转一下,放大一下,很恶心.于是就需要执行脚本来变换.

注意: 执行不同的顺序的变换,会导致完全不同的结果.





各位雷神,可以睁大你的硬化氪金眼, 区别有木有啊!!

## 好吧,下面讲如何创建脚本

Creating Affine Transforms; 老湿给您介绍霞,affine是仿射,联姻,投影变换的意思. 说白了,这句话就是生成一个几何的变换脚本

下面看,就是简单的3组创建,应用的脚本.

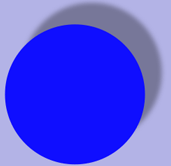
**Table 5-1**  Affine transform functions for translation, rotation, and scaling

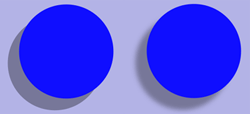
|  |  |
| --- | --- |
| **Function** | **Use** |
| CGAffineTransformMakeTranslation | To construct a new translation matrix from x and y values that specify how much to move the origin. |
| CGAffineTransformTranslate | To apply a translation operation to an existing affine transform. |
| CGAffineTransformMakeRotation | To construct a new rotation matrix from a value that specifies in radians how much to rotate the coordinate system. |
| CGAffineTransformRotate | To apply a rotation operation to an existing affine transform. |
| CGAffineTransformMakeScale | To construct a new scaling matrix from x and y values that specify how much to stretch or shrink coordinates. |
| CGAffineTransformScale | To apply a scaling operation to an existing affine transform. |

## 讲完这么多,我就不说Pattern,直接过来讲Shadows阴影技术吧

第一个图讲阴影

三个参数,offset即阴影的偏离值,这个又分两种,显然是X,Y方向的偏离啦.Y是指纵坐标,离用户视角的距离. X是指, 横向偏移数值.还有最后一个参数, blur, 模拟度啦.可以清晰,可以模糊.





API如下

CGContextSetShadow,

提供graphics context 标本,提供offsets,blur.即可.

你也可以用如下API设定阴影的颜色.

CGContextSetShadowWithColor

下面代码解释了如何绘图弄出来阴影

|  |
| --- |
| void MyDrawWithShadows (CGContextRef myContext,  // 1 |
| float wd, float ht); |
| { |
| CGSize myShadowOffset = CGSizeMake (-15, 20);  // 2 |
| float myColorValues[] = {1, 0, 0, .6};  // 3 颜色数组 |
| CGColorRef myColor;  // 4 |
| CGColorSpaceRef myColorSpace;  // 5 |
|  |
| CGContextSaveGState(myContext);  // 6 保存一下当前context状态 |
|  |
| CGContextSetShadow (myContext, myShadowOffset, 5);  // 7 重点,上阴影 |
| // Your drawing code here  // 8 |
| CGContextSetRGBFillColor (myContext, 0, 1, 0, 1); |
| CGContextFillRect (myContext, CGRectMake (wd/3 + 75, ht/2 , wd/4, ht/4)); |
|  |
| myColorSpace = CGColorSpaceCreateDeviceRGB ();  // 9 |
| myColor = CGColorCreate (myColorSpace, myColorValues);  // 10 生成一个colorSpace对象,提供刚创建的空间,以及颜色数组,注意是RGBA格式. |
| CGContextSetShadowWithColor (myContext, myShadowOffset, 5, myColor);  // 11 |
| // Your drawing code here  // 12 |
| CGContextSetRGBFillColor (myContext, 0, 0, 1, 1); |
| CGContextFillRect (myContext, CGRectMake (wd/3-75,ht/2-100,wd/4,ht/4)); |
|  |
| CGColorRelease (myColor);  // 13 |
| CGColorSpaceRelease (myColorSpace);  // 14 |
|  |
| CGContextRestoreGState(myContext);  // 15 |
| } |

好吧,阴影解释完毕.今儿翻译到此为止.