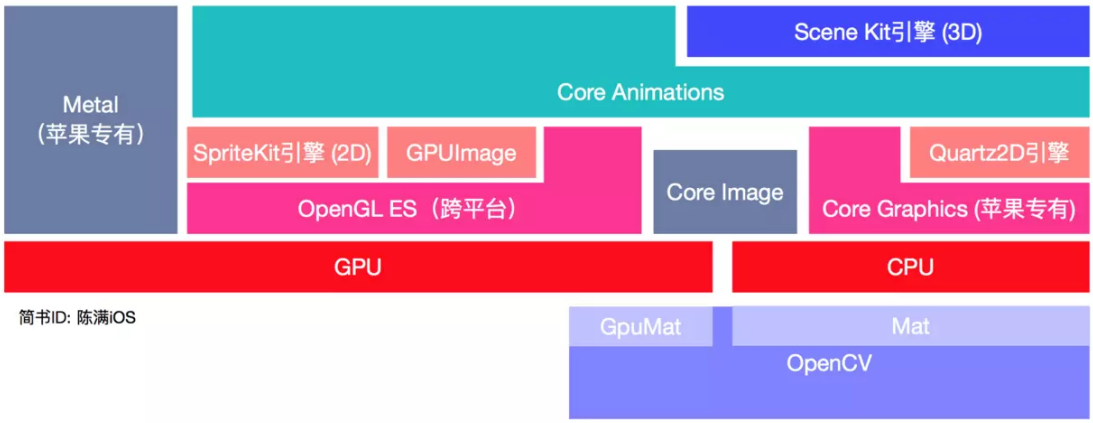
前言

对于刚接触iOS图形相关框架的小白，有一些图形框架在**字面**上和**功能**上非常容易混淆。这里旨在总结一下各种框架，区分它们的概念和功能，以作日后进一步细分学习的指引。因而，**本文并不会针对具体框架作详解，只作区分引导**，读者可自行选择方向继续深造，笔者日后也可能抽空产出这一块的文章。为此，笔者总结了一张各种框架关系图，如下所示：



总的来说，iOS与图形图像处理相关的框架都在这里了：

- i. 界面图形框架 -- UIKit
- ii. 核心动画框架 -- Core Animation
- iii. 苹果封装的图形框架 -- Core Graphics & Quartz 2D
- iv. 传统跨平台图形框架 -- OpenGL ES
- v. 苹果最新力推的图形框架 -- Metal
- vi. 适合图片的苹果滤镜框架 -- Core Image
- vii. 适合视频的第三方滤镜方案 -- GPUImage
- viii. 游戏引擎 -- Scene Kit (3D) 和 Sprite Kit (2D)
- ix. 计算机视觉在iOS的应用 -- OpenCV for iOS

1. 界面图形框架 -- UIKit

UIKit

UIKit是一组Objective-C API，为线条图形、Quartz图像和颜色操作提供Objective-C 封装，并提供2D绘制、图像处理及用户接口级别的动画。

UIKit包括UIBezierPath（绘制线、角度、椭圆及其它图形）、UIImage（显示图像）、UIColor（颜色操作）、UIFont和UIScreen（提供字体和屏幕信息）等类以及在位图图形环境、PDF图形环境上进行绘制和 操作的功能等, 也提供对标准视图的支持，也提供对打印功能的支持。

UIKit与Core Graphics的关系

在UIKit中，UIView类本身在绘制时**自动创建一个图形环境，即Core Graphics层的CGContext类型，作为当前的图形绘制环境**。在绘制时可以调用 `UIGraphicsGetCurrentContext` 函数获得当前的图形环境，例如：

```
- (void)drawRect:(CGRect)rect {
    // Drawing code
    NSLog(@"%s",__func__);
    //1.获取上下文
    CGContextRef contextRef = UIGraphicsGetCurrentContext();
    //2.描述路径
    UIBezierPath * path = [UIBezierPath bezierPath];
    //起点
    [path moveToPoint:CGPointMake(10, 10)];
    //终点
    [path addLineToPoint:CGPointMake(100, 100)];
    //设置颜色
    [[UIColor whiteColor]setStroke];
    //3.添加路径
    CGContextAddPath(contextRef, path.CGPath);
    //显示路径
    CGContextStrokePath(contextRef);

}
复制代码
```

这段代码就是在UIView的子类中调用 `UIGraphicsGetCurrentContext` 函数获得当前的图形环境，然后向该图形环境添加路径，最后绘制。

2. 核心动画框架 -- Core Animation

Core Animation

Core Animation 是一套Objective-C API，实现了一个高性能的复合引擎，并提供一个简单易用的编程接口，给用户UI添加平滑运动和动态反馈能力。

Core Animation 是 UIKit 实现动画和变换的基础，也负责视图的复合功能。使用Core Animation可以实现定制动画和细粒度的动画控制，创建复杂的、支持动画和变换的layered 2D视图。

Core Animation 不属于绘制系统，但它是以硬件复合和操作显示内容的基础设施。这个基础设施的核心是layer对象，用来管理和操作显示内容。在 iOS 中 **每一个视图都对应Core Animation的一个层对象**，与视图一样，层之间也组织为层关系树。一个层捕获视图内容为一个被图像硬件容易操作的位图。在多数应用中层作为管理视图的方式使用，但也可以创建独立的层到一个层关系树中来显示视图不够支持的显示内容。

OpenGL ES的内容也可以与Core Animation内容进行集成。

为了使用Core Animation实现动画，可以修改 **层的属性值** 来触发一个action对象的执行，不同的action对象实现不同的动画。

Core Animatio相关基类及子类

Core Animation 提供了一下一组应用可以采用的类来提供对不同动画类型的支持：

- **CAAnimation** 是一个抽象公共基类，CAAnimation采用CAMediaTiming 和CAAction协议为动画提供时间（如周期、速度、重复次数等）和action行为（启动、停止等）。
- **CAPropertyAnimation** 是 CAAnimation的抽象子类，为动画提供一个由一个key路径规定的层属性的支持；
- **CABasicAnimation** 是CAPropertyAnimation的具体子类，为一个层属性提供简单插入能力。
- **CAKeyframeAnimation** 也是CAPropertyAnimation的具体子类，提供key帧动画支持。

3. 苹果封装的图形框架 -- Core Graphics & Quartz 2D

Core Graphics

Core Graphics是一套C-based API， 支持向量图形，线、形状、图案、路径、剃度、位图图像和pdf 内容的绘制。

Quartz 2D

Quartz 2D是Core Graphics中的2D 绘制呈现引擎。Quartz是资源和设备无关的,提供路径绘制，anti-aliased呈现，剃度填充图案，图像，透明绘制和透明层、遮蔽和阴影、颜色管理，坐标转换，字体、offscreen呈现、pdf文档创建、显示和分析等功能。

Quartz 2D能够与所有的图形和动画技术（如Core Animation, OpenGL ES, 和 UIKit 等）一起使用。

Quartz 2D采用paint模式进行绘制。

图形环境Context

Quartz 2D中使用的图形环境也由一个类CGContext表示。

在Quartz 2D中可以把一个图形环境作为一个绘制目标。当使用Quartz 2D进行绘制时，所有设备特定的特性被包含在你使用的特定类型的图形环境中，因此通过给相同的图像操作函数提供不同的图像环境你就能够画相同的图像到不同的设备上，因此做到了图像绘制的设备无关性。

图形环境Context是个比较抽象的东西，它不仅仅是一个可以绘制的图层，还包含为当前图层设置的参数，如阴影，线条粗细，绘制模式等。可以类比成一个新建的Photoshop图层以及当前笔触，颜色等配置。

对于移动平台，有三种常见的图形环境Context：

- 位图上下文（A bitmap graphics context）：一般用于绘制图片或者自定义控件。
    - View Graphics Context: 由UIView自动创建，你重写UIView drawRect方法时，你的内容会画在这个上下文上。
    - Bitmap Graphics Context: 绘制在该上下文的内容会以点阵形式存储在一块内存中。简单说，就是为图片开辟一块内存，然后在里面画东西，上下文帮你把图片内存抽象成一个Context(图层)了。
- PDF上下文（A PDF graphics context）：用于生成pdf文件。
- 图层上下文（A layer context）：用于离屏绘制（ offscreen drawing）。

Quartz 2D提供的主要类包括：

- CGContext：表示一个图形环境；
- CGPath：使用向量图形来创建路径，并能够填充和stroke；
- CGImage：用来表示位图；
- CGLayer：用来表示一个能够用于重复绘制和offscreen绘制的绘制层；
- CGPattern：用来表示Pattern，用于重复绘制；
- CGShading和 CGGradient：用于绘制剃度；
- CGColor 和 CGColorSpace；用来进行颜色和颜色空间管理；
- CGFont, 用于绘制文本；
- CGPDFContentStream、CGPDFScanner、CGPDFPage、CGPDFObject,CGPDFStream, CGPDFString等用来进行pdf文件的创建、解析和显示。

4. 传统跨平台图形框架 -- OpenGL ES

OpenGL ES

OpenGL ES是一套多功能开放标准的用于嵌入系统的C-based的图形库，用于2D和3D数据的可视化。OpenGL被设计用来转换一组图形调用功能到底层图形硬件（GPU），由GPU执行图形命令，用来实现复杂的图形操作和运算，从而能够高性能、高帧率利用GPU提供的2D和3D绘制能力。

OpenGL ES规范本身不定义绘制表面和绘制窗口，因此ios为了使用它必须提供和创建一个OpenGL ES 的呈现环境，创建和配置存储绘制命令结果的framebuffer 及创建和配置一个或多个呈现目标。

EAGL

在 iOS中使用EAGL提供的EAGLContext类 来实现和提供一个呈现环境，用来保持OpenGL ES使用到的硬件状态。 EAGL是一个Objective-C API，提供使OpenGL ES与Core Animation和UIKIT集成的接口。

在调用任何OpenGL ES 功能之前必须首先初始化一个EAGLContext 对象。每一个IOS应用的每一个线程都有一个当前context，在调用OpenGL ES函数时，使用或改变此context中的状态。

EAGLContext 的类方法setCurrentContext: 用来设置当前线程的当前context。EAGLContext 的类方法currentContext 返回当前线程的当前context。在切换相同线程的两个上下文之前，必须调用glFlush函数来确保先前已提交的命令被提交到图形硬件中。

GLKit

可以采用不同的方式使用OpenGL ES以便呈现OpenGL ES内容到不同的目标：GLKit和CAEAGLLayer。

为了创建全屏幕的视图或使OpenGL ES内容与UIKit视图集成，可以使用GLKit。在使用GLKit时，GLKit提供的类GLKView类本身实现呈现目标及创建和维护一个framebuffer。

GLKit是一组Objective-C 类，为使用OpenGL ES 提供一个面向对象接口，用来简化OpenGL ES应用的开发。

CAEAGLLayer

为了使OpenGL ES内容作为一个Core Animation层的部分内容时，可以使用CAEAGLLayer 作为呈现目标，并需要另外创建framebuffer以及自己实现和控制整个绘制流程。

GLKit支持四个3D应用开发的关键领域：

1） GLKView 和GLKViewController类提供一个标准的OpenGL ES视图和相关联的呈现循环。GLKView可以作为OpenGL ES内容的呈现目标，GLKViewController提供内容呈现的控制和动画。视图管理和维护一个framebuffer，应用只需在framebuffer进行绘画即可。

2）GLKTextureLoader 为应用提供从IOS支持的各种图像格式的源自动加载纹理图像到OpenGL ES 图像环境的方式，并能够进行适当的转换，并支持同步和异步加载方式。

3）数学运算库，提供向量、矩阵、四元数的实现和矩阵堆栈操作等OpenGL ES 1.1功能。

4）Effect效果类提供标准的公共着色效果的实现。能够配置效果和相关的顶点数据，然后创建和加载适当的着色器。GLKit 包括三个可配置着色效果类：GLKBaseEffect实现OpenGL ES 1.1规范中的关键的灯光和材料模式，GLKSkyboxEffect提供一个skybox效果的实现, GLKReflectionMapEffect 在GLKBaseEffect基础上包括反射映射支持。

5. 苹果最新力推的图形框架 -- Metal

Metal框架支持GPU硬件加速、高级3D图形渲染以及大数据并行运算。且提供了先

进而精简的API来确保框架的细粒度(fine-grain)，并且在组织架构、程序处理、图形呈现、运算指令以及指令相关数据资源的管理上都支持底层控制。其核心目的是尽可能的减少CPU开销，而将运行时产生的大部分负载交由GPU承担

编写基于底层图形 API 的渲染引擎时，除了 Metal 以外的其他选择还有 OpenGL 和 OpenGL ES。OpenGL 不仅支持包括 OSX，Windows，Linux 和 Android 在内的几乎所有平台，还有大量的教程，书籍和最佳实践指南等资料。目前，Metal 的资源非常有限，并且仅限于搭载了 64 位处理器的 iPhone 和 iPad。但另外一方面，因为 OpenGL 的限制，其性能与 Metal 相比并不占优势，毕竟后者是专门用来解决这些问题的。

如果想要一个 iOS 上高性能的并行计算库，答案非常简单。Metal 是唯一的选择。OpenGL 在 iOS 上是私有框架，而 Core Image (使用了 OpenGL) 对这样的任务来说既不够强大又不够灵活。

6. 适合图片的苹果滤镜框架 -- Core Image

Core Image 是 iOS5 新加入到 iOS 平台的一个图像处理框架，提供了强大高效的图像处理功能， 用来对基于像素的图像进行操作与分析， 内置了很多强大的滤镜(Filter)(目前数量超过了180种)， 这些Filter 提供了各种各样的效果， 并且还可以通过 `滤镜链` 将各种效果的 `Filter叠加` 起来形成强大的自定义效果。

一个 **滤镜** 是一个对象，有很多输入和输出，并执行一些变换。例如，模糊滤镜可能需要输入图像和一个模糊半径来产生适当的模糊后的输出图像。

一个 **滤镜链** 是一个链接在一起的滤镜网络，使得一个滤镜的输出可以是另一个滤镜的输入。以这种方式，可以实现精心制作的效果。

iOS8 之后更是支持自定义 CIFilter，可以定制满足业务需求的复杂效果。

Core Image 的 API 主要就是三类：

- CIImage 保存图像数据的类，可以通过UIImage，图像文件或者像素数据来创建，包括未处理的像素数据。
- CIFilter 表示应用的滤镜，这个框架中对图片属性进行细节处理的类。它对所有的像素进行操作，用一些键-值设置来决定具体操作的程度。
- CIContext 表示上下文，如 Core Graphics 以及 Core Data 中的上下文用于处理绘制渲染以及处理托管对象一样，Core Image 的上下文也是实现对图像处理的具体对象。可以从其中取得图片的信息。

Core Image 的另外一个优势，就是可以根据需求选择 CPU 或者 GPU 来处理。

```
// 创建基于 CPU 的 CIContext 对象 (默认是基于 GPU, CPU 需要额外设置参数)
context = [CIContext contextWithOptions: [NSDictionary
dictionaryWithObject:[NSNumber numberWithBool:YES]
forKey:kCIContextUseSoftwareRenderer]];

// 创建基于 GPU 的 CIContext 对象
context = [CIContext contextWithOptions: nil];

// 创建基于 GPU 的 CIContext 对象
EAGLContext *eaglctx = [[EAGLContext alloc]
initWithAPI:kEAGLRenderingAPIOpenGLES2];
context = [CIContext contextWithEAGLContext:eaglctx];
复制代码
```

7. 适合视频的第三方滤镜方案 -- GPUImage

GPUImage 优势： 最低支持 iOS 4.0，iOS 5.0 之后就支持自定义滤镜。 在低端机型上，GPUImage 有更好的表现。（这个没用真正的设备对比过，GPUImage 的主页上是这么说的） GPUImage 在视频处理上有更好的表现。 GPUImage 的代码完成公开，实现透明。 可以根据自己的业务需求，定制更加复杂的管线操作。可定制程度高。

8. 游戏引擎 -- Scene Kit (3D) 和 Sprite Kit (2D)

对于寻找游戏引擎的开发者来说，Metal 不是最佳选择。苹果官方的的 Scene Kit (3D) 和 Sprite Kit (2D) 是更好的选择。这些 API 提供了包括物理模拟在内的更高级别的游戏引擎。

另外还有功能更全面的 3D 引擎，例如 Epic 的 Unreal Engine 或 Unity，二者都是跨平台的。使用这些引擎，你无需直接使用 Metal 的 API，就可以从 Metal 中获益。

2D渲染 -- SpriteKit

SpriteKit 让开发者可以开发高性能、省电节能的 2D 游戏。在 iOS 8 中，新添了多项增强功能，这将使 2D 游戏体验更加精彩。这些新技术有助于使游戏角色的动作更加自然，并让开发者可以更轻松地在游戏中加入力场、检测碰撞和生成新的灯光效果。

3D渲染 -- SceneKit

SceneKit 专为休闲 3D 游戏而设计，可让开发者渲染 3D 游戏场景。SceneKit 内置了物理引擎、粒子发生器和各种易用工具，可以轻松快捷地为 3D 物体编写动作。不仅如此，它还与 SpriteKit 完全集成，所以开发者可以直接在 3D 游戏中加入 SpriteKit 的素材。

9. 计算机视觉在iOS的应用 -- OpenCV for iOS

OpenCV 的 API 是 C++ 的。它由不同的模块组成，这些模块中包含范围极为广泛的各种方法，从底层的图像颜色空间转换到高层的机器学习工具。这里提供一个入门PDF文档 下载入口。

使用 C++ API 并不是绝大多数 iOS 开发者每天都做的事，你需要使用 Objective-C++ 文件来调用 OpenCV 的函数。 也就是说，你不能在 Swift 或者 Objective-C 语言内调用 OpenCV 的函数。 这篇 OpenCV 的 iOS 教程告诉你只要把所有用到 OpenCV 的类的文件后缀名改为 `.mm` 就行了，包括视图控制器类也是如此。这么干或许能行得通，却不是什么好主意。正确的方式是给所有你要在 app 中使用到的 OpenCV 功能写一层 Objective-C++ 封装。这些 Objective-C++ 封装把 OpenCV 的 C++ API 转化为安全的 Objective-C API，以方便地在所有 Objective-C 类中使用。

走封装的路子，你的工程中就可以只在这些封装中调用 C++ 代码，从而避免掉很多让人头痛的问题，比如直接改文件后缀名会因为在错误的文件中引用了一个 C++ 头文件而产生难以追踪的编译错误。

OpenCV 声明了命名空间 `cv`，因此 OpenCV 的类的前面会有个 `cv::` 前缀，就像 `cv::Mat`、 `cv::Algorithm` 等等。你也可以在 `.mm` 文件中使用 `using namespace cv` 来避免在一堆类名前使用 `cv::` 前缀。但是，在某些类名前你必须使用命名空间前缀，比如 `cv::Rect` 和 `cv::Point`，因为它们会跟定义在 `MacTypes.h` 中的 `Rect` 和 `Point` 相冲突。尽管这只是个人偏好问题，我还是偏向在任何地方都使用 `cv::` 以保持一致性。

一般讲的OpenCV是基于CPU的，相关资料和支持也是最完善的。当然，也有基于GPU模块，但提供的接口非常坑爹，相当一部分不支持浮点类型（像histogram、integral这类常用的都不支持）；又如，遇到阈值判断的地方，就必须传回cpu处理，因为gpu函数都是并行处理的，每改写完一个算法模块，就测试一下运行效率，有的时候是振奋人心，有的时候则是当头棒喝——比CPU还慢。详情可参阅 这里。

10. 参考文献

- Core Animations https://www.sohu.com/a/203987045_468740
  https://blog.csdn.net/huangznian/article/details/42919221
  https://www.jianshu.com/p/446a6b72f981
  https://www.jianshu.com/p/439e158b44de

- Metal https://juejin.im/post/59ad5d6551882539255b4809
  https://www.jianshu.com/p/ce53d0178f20
  https://blog.csdn.net/pizi0475/article/details/50232029
  https://baike.baidu.com/item/Metal/10917053?fr=aladdin
  https://zhuanlan.zhihu.com/p/24623380?
  utm_source=tuicool&utm_medium=referral

- Core Image https://objccn.io/issue-21-6/
  http://colin1994.github.io/2016/10/21/Core-Image-OverView/?
  utm_source=tuicool&utm_medium=referral
  https://blog.csdn.net/jingcheng345413/article/details/54967640
  https://www.cnblogs.com/try2do-neo/p/3601546.html

- Core Graphics https://www.jianshu.com/p/e7a50dcbe7c8
  https://www.jianshu.com/p/55cc1587e618
  https://www.jianshu.com/p/494c57f49479
  https://my.oschina.net/flyfishbay/blog/1504698

- OpenCV https://blog.csdn.net/zhonggaorong/article/details/78191514
  http://www.opencv.org.cn/forum.php?mod=viewthread&tid=33549
  https://blog.csdn.net/kelvin_yan/article/details/41804357
  https://blog.csdn.net/sinat_31135199/article/details/53053188
  https://blog.csdn.net/liyuefeilong/article/details/46292339

- GPUImage https://blog.csdn.net/fanbird2008/article/details/51707430

- 其它 https://blog.csdn.net/goohong/article/details/40743883