Scene Kit

objccn.io (http://getpocket.com/redirect?url=http%3A%2F%2Fobjccn.io%2Fissue-18-3%2F)

查看原始文档

在 WWDC 2012 (那时 OS X 系统还在用喵系命名)，Apple 向 OS X 开发者们介绍了 Scene Kit，这个 Cocoa 下的 3D 渲染框架。在第一版通用 3D 渲染器发布后，一年内又陆续增加了像 shader (着色器) 修改器、节点约束、骨骼动画等几个强大的特性 (随 Mavericks 发布)。今年，Scene Kit 变的更加强大，支持了粒子效果、物理引擎、脚本事件以及多通道分层渲染等多种技术，而且，对于很多人来说更关键的是，它终于可以在 iOS 中使用了。

一上手，我发现 Scene Kit 最强大和脱颖而出的地方，就是可以与 Core Image，Core Animation，Sprite Kit 等已有的图形框架相互整合及协作，这在其他游戏引擎中可不常见，但如果你本身就是个 Cocoa 或 Cocoa Touch 框架下的的开发者的话，就会感到相当亲切了。

Scene Kit 概要

Scene Kit 建立在 OpenGL 的基础上，包含了如光照、模型、材质、摄像机等高级引擎特性，这些组件都是面向对象的，你可以用熟悉的 Objective-C 或 Swift 语言来编写代码。假如你用过 OpenGL 最早的版本 (https://www.opengl.org/documentation/specs/version1.1/glspec1.1/node30.html#SEC 那时还没有 shader，只能苦逼的使用各种底层受限制的 API 开发。而 Scene Kit 就好了很多，对于大多数需求 (甚至像动态阴影和景深 (https://en.wikipedia.org/wiki/Depth_of_field)这种高级特性)，使用它提供的上层 API 来配置，就已经足够了。

不仅如此，Scene Kit 还允许你直接调用底层 API，或自己写 shader 进行手动渲染 (GLSL (https://en.wikipedia.org/wiki/OpenGL_Shading_Language))。

节点 (Nodes)

不仅是光照、模型、材质、摄像机这几个具体的对象，Scene Kit 使用节点 (在3D图形学中，像这样的树状节点结构一般被称做 *scene graph*，这也是 Scene Kit 名称由来的一种解释) 以树状结构来组织内容，每个节点都存储了相对其父节点的位移、旋转角度、缩放等信息，父节点也是如此，一直向上，直到根节点。假如要给一个节点确定一个位置，就必须将它挂载到节点树中的某个节点上，可以使用下面的几个操作方法：

- **`addChildNode(_:)`**
- **`insertChildNode(_: atIndex:)`**
- **`removeFromParentNode()`**

这些方法与 iOS 和 OS X 中管理 view 和 layer 层级方法如出一辙。

几何模型对象

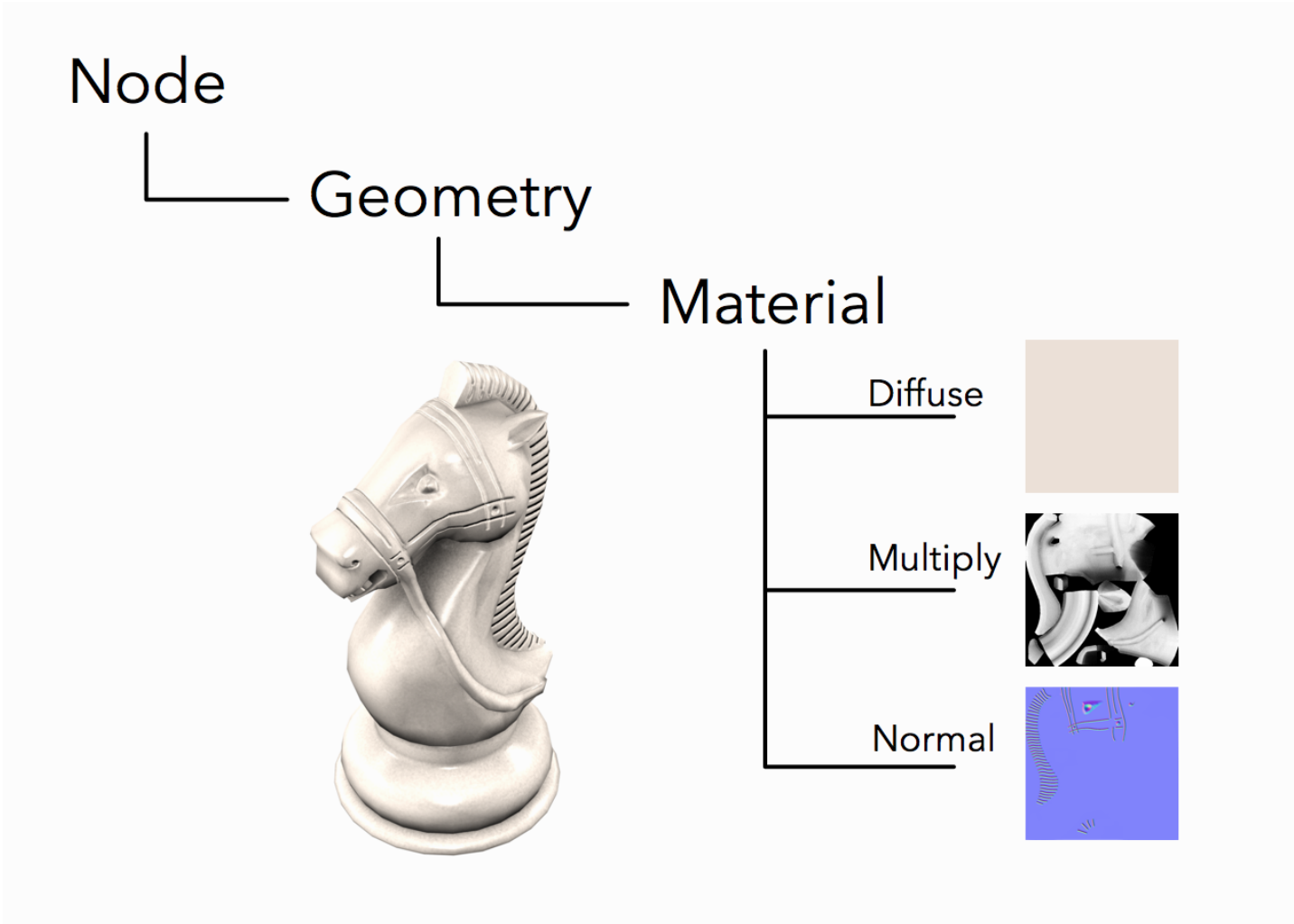
Scene Kit 内建了几种简单的几何模型，如盒子、球体、平面、圆锥体等，但对于游戏来说，一般都会从文件中加载3D模型。你可以通过制定文件名来导入 (或导出) COLLADA (https://en.wikipedia.org/wiki/COLLADA) 格式的模型文件：

```
let chessPieces = SCNScene(named: "chess pieces") //
SCNScene?
```

如果一个从文件里加载的场景可以全部显示时，将其设置成 SCNView 的 scene 就好了；但如果加载的场景文件中包含了多个对象，只有一部分对象要显示在屏幕上时，就可以通过名字找到这个对象，再手动加载到 view 上：

```
if let knight =
chessPieces.rootNode.childNodeWithName("Knight",
recursively: true) {
    sceneView.scene?.rootNode.addChildNode(knight)
}
```

这是一个对导入文件原始节点的引用，其中包含了任一和每一个子节点，也包括了模型对象 (包括其材质)，光照，以及绑定在这些节点上的摄像机。只要传入的名字一样，不论调用多少次，返回的都是对同一个对象的引用。



若需要在场景中拥有一个节点的多个拷贝，如在一个国际象棋棋盘上显示两个马，你可以对马这个节点进行 **copy** 或 **clone** (递归的copy)。这将会拷贝一份节点的引用，但两份引用所指向的材质对象和模型对象仍然是原来那个。所以，想要单独改变副本材质的话，需要再copy一份模型对象，并对这个新的模型对象设置新材质。copy一个模型对象的速度仍然很快，开销也不高，因为副本引用的顶点数据还是同一份。

带有骨骼动画的模型对象也会拥有一个皮肤对象，它提供了对骨骼中各个节点的访问接口，以及管理骨骼和模型间连接的功能。每个单独的骨骼都可以被移动和旋转，而复杂的动画需要同时对多块骨骼进行操作，如一个角色走路的动画，很可能就是从文件读取并加到对象上的 (而不是用代码一根骨头一根骨头的写)。

光照

Scene Kit 中完全都是动态光照，使用起来一般会很简单，但也意味着与完整的游戏引擎相比，光照这块进步并不明显。Scene Kit 提供四种类型的光照：环境光、定向光源、点光源和聚光灯。

通常来说，旋转坐标轴和变换角度并不是设定光照的最佳方法。下面的例子表示一个光照对象通过一个节点对象来设置空间坐标，再通过 "look at" 约束，将光照对象约束到了目标对象上，即使它移动，光照也会一直朝向目标对象。

```
let spot = SCNLight()
spot.type = SCNLightTypeSpot
spot.castsShadow = true

let spotNode = SCNNode()
spotNode.light = spot
spotNode.position = SCNVector3(x: 4, y: 7, z: 6)

let lookAt = SCNLookAtConstraint(target: knight)
spotNode.constraints = [lookAt]
```

动画

Scene Kit 的对象中绝大多数属性都是可以进行动画的，就像 Cocoa (或 Cocoa Touch) 框架一样，你可以创建一个 **CAAnimation** 对象，并指定一个 key path (甚至可以 **"position.x"**)，然后向一个对象施加这个动画。同样的，你可以在

SCNTransaction 的 "begin" 和 "commit" 调用间去改变值，和刚才的 CAAnimation 非常相似：

```
let move = CABasicAnimation(keyPath: "position.x")
move.byValue  = 10
move.duration = 1.0
knight.addAnimation(move, forKey: "slide right")
```

Scene Kit 也提供了像 Sprite Kit 那样的 action 形式的动画 API，你可以创建串行的动画组，也支持自定义 action 来协同使用。与 Core Animation 不同的是，这些 action 作为游戏循环的一部分执行，在每一帧都更新模型对象的值，而不只是更新表现层的节点。

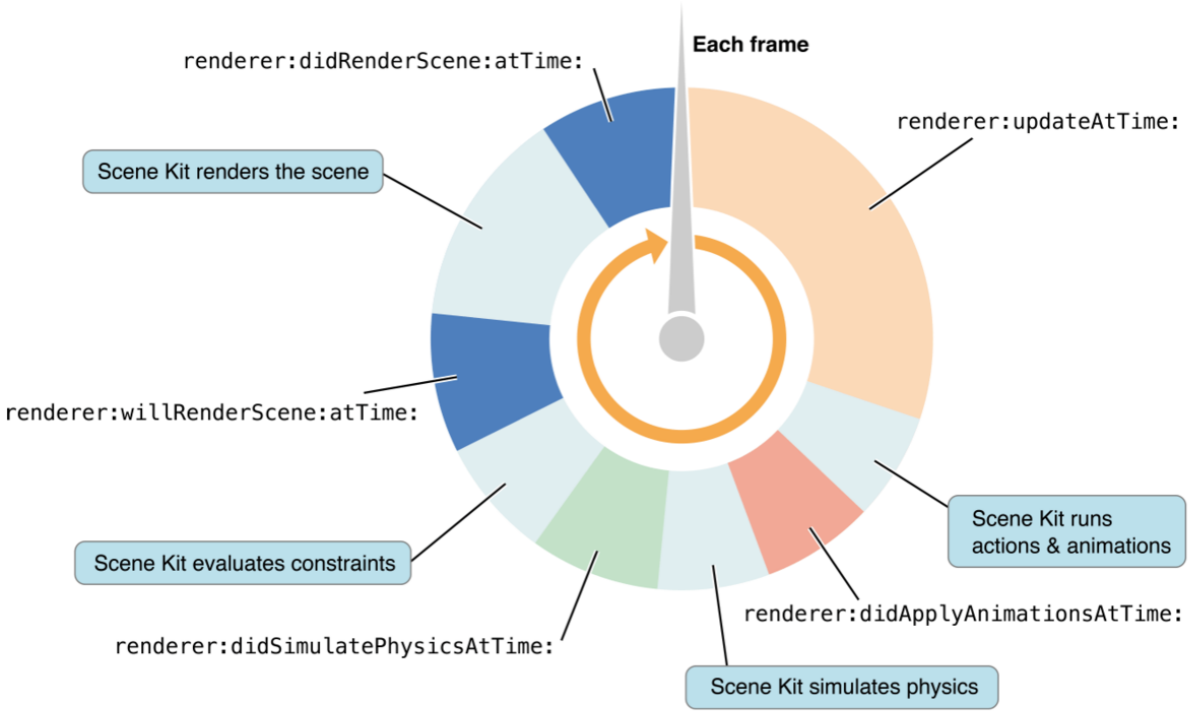
假如你之前用过 Sprite Kit，会发现 Scene Kit 除了变成了 3D 之外，没有太多陌生的东西。目前，在 iOS8 (首次支持 Scene Kit) 和 OS X 10.10 下，Scene Kit 和 Sprite Kit 可以协同工作：对 Sprite Kit 来说，3D 模型可以与 2D 精灵混合使用；对 Scene Kit 来说，Sprite Kit 中的场景和纹理可以作为 Scene Kit 的纹理贴图，而且 Sprite Kit 的场景可以作为 Scene Kit 场景的蒙层 (如3D游戏中的2D菜单面板，译者注。两套非常像的API和概念 (像场景啊，节点啊，约束啊两边都有) 让人容易混淆。)

开始用 Scene Kit 写游戏

不仅是动作和纹理，Scene Kit 和 Sprite Kit 还有很多相同之处。当开始写游戏的时候，Scene Kit 和它 2D 版本的小伙伴非常相似，它们的游戏循环步骤完全一致，使用下面几个代理回调：

1. 更新场景
2. 应用动画/动作
3. 模拟物理效果
4. 应用约束
5. 渲染

Figure 1 Frame processing in a scene



这些回调在每帧被调用，并用来执行游戏相关的逻辑，如用户输入，AI (人工智能) 和游戏脚本。

处理用户输入

Scene Kit 与普通 Cocoa 或 Cocoa Touch 应用使用一样的机制来处理用户输入，如键盘事件、鼠标事件、触摸事件和手势识别，而主要区别在于 Scene Kit 中只有一个视图，场景视图 (scene view) 。像键盘事件或如捏取、滑动、旋转的手势，只要知道事件的发生就好了，但像鼠标点击，或触碰、拖动手势等就需要知道具体的事件信息了。

这些情况下，scene view 可以使用 **`-hitTest(_: options:)`** 来做点击测试。与通常的视图只返回被点击的子 view 或子 layer 不同，Scene Kit 返回一个数组，里面存有每个相交的模型对象以及从摄像机投向这个测试点的射线。每个 hit test 的结果包含被击中模型的节点对象，也包含了交点的详细信息 (交点坐标、交点表面法线，交点的纹理坐标)。多数情况下，知道第一个被击中的节点就足够了：

```
if let firstHit = sceneView.hitTest(tapLocation, options:
nil)?.first as? SCNHitTestResult  {
    let hitNode = firstHit.node
    // do something with the node that was hit...
}
```

扩展默认渲染流程

光照和材质的配置方法很易用，但有局限性。假如你有写好的 OpenGL 着色器 (shader)，可以用于完全自定制的进行材质渲染；如果你只想修改下默认的渲染，Scene Kit 暴露了 4 个入口用于插入 shader代码 (GLSL) 来改变默认渲染。Scene Kit 在不同入口点分别提供了对旋转矩阵、模型数据、样本贴图及渲染后输出的色值的访问。

比如，下面的 GLSL 代码被用在模型数据的入口点中，可以将模型对象上所有点沿 x 轴扭曲。这是通过定义一个函数来创建一个旋转变换，并将其应用在模型的位置和法线上。同时，也自定义了一个 "uniform" 变量 (https://www.opengl.org/wiki/Uniform_)来决定对象该如何被扭曲。

```
// a function that creates a rotation transform matrix
around X
mat4 rotationAroundX(float angle)
{
    return mat4(1.0,    0.0,         0.0,        0.0,
                0.0,    cos(angle), -sin(angle), 0.0,
                0.0,    sin(angle),  cos(angle), 0.0,
                0.0,    0.0,         0.0,        1.0);
}

#pragma body

uniform float twistFactor = 1.0;
float rotationAngle = _geometry.position.x * twistFactor;
mat4 rotationMatrix = rotationAroundX(rotationAngle);

// position is a vec4
_geometry.position *= rotationMatrix;

// normal is a vec3
vec4 twistedNormal = vec4(_geometry.normal, 1.0) *
rotationMatrix;
_geometry.normal   = twistedNormal.xyz;
```

着色修改器 (Shader modifier) 既可以绑定在模型对象上，也可以绑定在它的材质对象上。这两个类都完全支持 key-value coding (KVC)，你可以指定任意 key 进行赋值。在 shader 中声明的 "twistFactor" uniform 变量使得 Scene Kit 在这个值改变时自动重新绑定 uniform，这使得你也可以用 KVC 来实现：

```
torus.setValue(5.0, forKey: "twistFactor")
```

使用这个 key path 的 **CAAnimation** 也 ok：

```
let twist = CABasicAnimation(keyPath: "twistFactor")
twist.fromValue = 5
twist.toValue   = 0
twist.duration  = 2.0

torus.addAnimation(twist, forKey: "Twist the torus")
```

延时着色

即使在纯 OpenGL 环境下，有些图像效果也无法通过一次渲染 pass 完成，我们可以将不同 shader 进行序列操作，以达到后续处理的目的，称为延时着色 (https://en.wikipedia.org/wiki/Deferred_shading)。Scene Kit 使用 SCNTechnique 类 (https://developer.apple.com/library/ios/documentation/SceneKit/Reference/SCNTechn 来表示这种技术。它使用字典来创建，字典中定义了绘图步骤、输入输出、shader 文件、符号等等。

第一个渲染 pass 永远是 Scene Kit 的默认渲染，它输出场景的颜色和景深。如果你不想这时计算色值，可以将材质设置成"恒定"的光照模型，或者将场景里所有光照都设置成环境光。

比如，从 Scene Kit 渲染流程的第一个 pass 获取景深，第二个获取法线，第三个对其执行边界检测，你即可以沿轮廓也可以沿边缘画粗线：

延伸阅读

如果你想了解更多使用 Scene Kit 做游戏的知识的话，我推荐今年的 WWDC 中的 "Building a Game with Scene Kit" video (https://developer.apple.com/videos/wwdc/2014/?id=610)，并看看 Bananas sample code (https://developer.apple.com/library/ios/samplecode/Bananas/Introduction/Intro.html)

如果你想学习 Scene Kit 基础知识，我推荐看这一年的 (https://developer.apple.com/videos/wwdc/2014/?id=609)和去年的 (https://developer.apple.com/videos/wwdc/2013/?id=500) "What's New in Scene Kit" 相关视频。如果你还想学习更多，可以关注我即将发布的这个主题的书 (http://scenekitbook.com)。