ARKit 文档翻译

Apple 原文地址：https://developer.apple.com/arkit/

iOS 11 引入了 ARKit，这是一个全新的框架，允许开发者轻松地为 iPhone 和 iPad 创建无与伦比的增强现实体验。通过将虚拟对象和虚拟信息同用户周围的环境相互融合，ARKit 使得应用跳出屏幕的限制，让它们能够以全新的方式与现实世界进行交互。

基础技术

视觉惯性里程计

ARKit 使用视觉惯性里程计 (Visual Inertial Odometry, VIO) 来精准追踪周围的世界。VIO 将摄像头的传感器数据同 Core Motion 数据进行融合。这两种数据允许设备能够高精度地感测设备在房间内的动作，而且无需额外校准。

场景识别与光亮估量

借助 ARKit，iPhone 和 iPad 可以分析相机界面中所呈现的场景，并在房间当中寻找水平面。ARKit 不仅可以检测诸如桌子和地板之类的水平面，还可以在较小特征点 (feature points) 上追踪和放置对象。ARKit 还利用摄像头传感器来估算场景当中的可见光总亮度，并为虚拟对象添加符合环境照明量的光量。

高性能硬件与渲染优化

ARKit 运行在 Apple A9 和 A10 处理器上。这些处理器能够为 ARKit 提供突破性的性能，从而可以实现快速场景识别，并且还可以让您基于现实世界场景，来构建详细并引人注目的虚拟内容。您可以利用 Metal、Scenekit 以及诸如 Unity、虚幻引擎之类的第三方工具，来对 ARKit 进行优化。

ARKit 概述

> ARKit 将 iOS 设备的摄像头和设备动作检测 (Motion) 功能，集成到您的应用或者游戏当中，从而为用户提供增强现实体验。

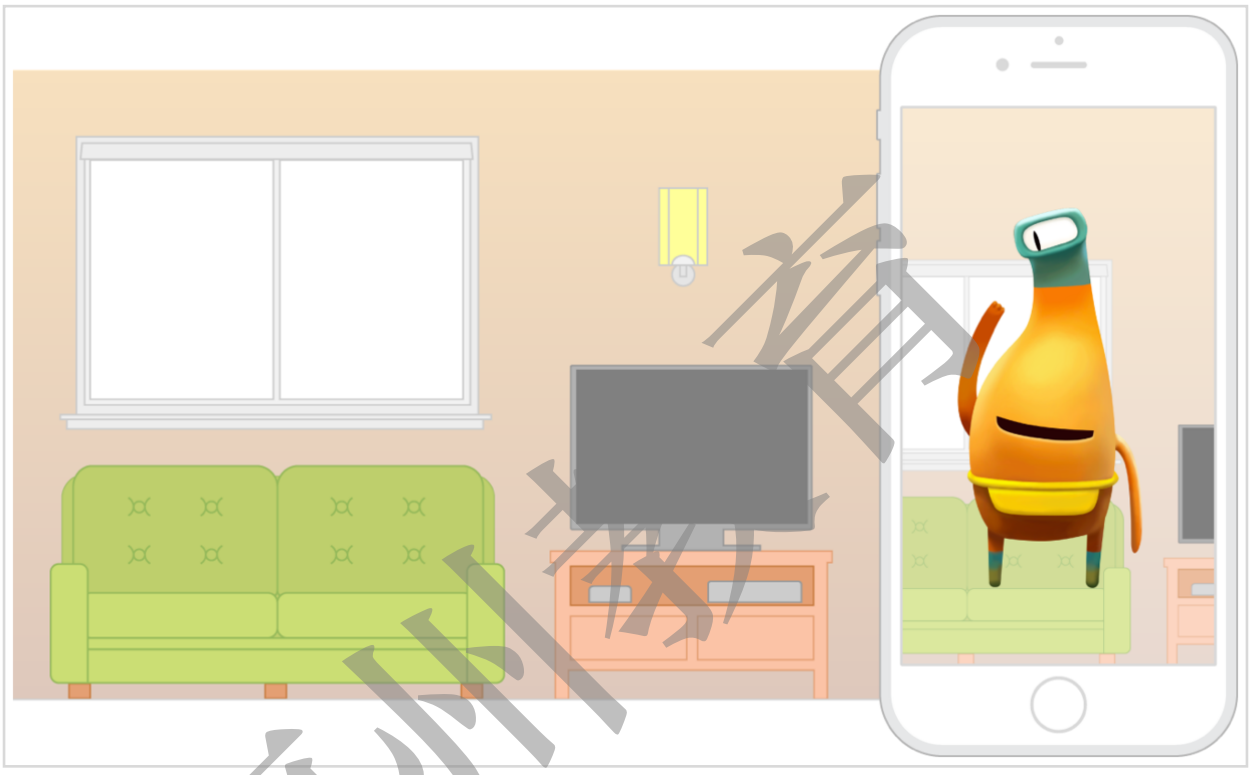
所谓的增强现实 (Augmented Reality, AR)，指的是向设备摄像头产生的实时动态视图中，添加 2D 或者 3D 元素，然后用某种方法让这些元素看起来就处于现实世界当中，所产生一种用户体验。ARKit 提供了设备动作追踪、相机场景捕获和高级场景处理，并让 AR 元素的展示变得极为便利，从而大大简化了建立 AR 用户体验的工作难度。

遵州教育

什么是增强现实？

探索 AR 的概念、特性，以及了解构建优秀 AR 场景的最佳实践。

概览



对于所有的 AR 场景而言，最基本要求是：创建并追踪现实空间和虚拟空间之间的关系，其中，现实空间是用户所处的世界，虚拟空间是可对可视化内容进行建模的世界，这同时也是 ARKit 的基本特征。当您的应用将这些虚拟内容与实时视频结合，并一起显示的时候，用户就可以体验到所谓的「增强现实」：您的虚拟内容成为了真实世界的一部分，尽管这只是「错觉」而已。

全局追踪是如何工作的

为了在现实世界和虚拟世界之间建立对应关系，ARKit 使用了一种被称为视觉惯性里程计的技术。这项技术会将 iOS 设备的动作感测硬件信息，加上对可见场景的计算机视觉分析功能，然后与设备的摄像头相结合。ARKit 将会去识别场景图像当中的显著特征，然后在视频帧中追踪这些特征位置的距离，然后再将这些信息与动作感测数据进行比较。从而生成具备设备位置和动作特征的高精度模型。

全局追踪 (World Tracking) 同样也可以分析和识别场景当中的内容。通过使用点击测试 (hit-testing) 方法（参见 ARHitTestResult 类），从而找到与相机图像中的某个点所对应的真实世界面。如果您在 Session (会话) 配置当中启用了 planeDetection 配置的话，那么 ARKit 就会去检测

相机图像当中的水平面，并报告其位置和大小。您可以使用点击测试所生成的结果，或者使用所检测到的水平面，从而就可以在场景当中放置虚拟内容，或者与之进行交互。

最佳实践与限制

全局追踪是一项不精确的科学 (inexact science)。尽管在这个过程当中，经常会产生可观的准确度，从而让 AR 的体验更加真实。然而，它严重依赖于设备物理环境的相关细节，而这些细节并不总是一致，有些时候也难以实时测量，这也就导致这些物理细节往往都会存在某种程度的错误。要建立高品质的 AR 体验，那么请注意下述这些注意事项和提示：

- 基于可见的照明条件来设计 AR 场景。全局追踪涉及到了图像分析的相关内容，因此就需要我们提供清晰的图像。如果摄像头没有办法看到相关的物理细节，比如说摄像头拍到的是一面空空如也的墙壁，或者场景的光线实在太暗的话，那么全局追踪的质量就会大大降低。

- 根据追踪质量的相关信息来给用户进行反馈提示。全局追踪会将图像分析与设备的动作模式关联起来。如果设备正在移动的话，那么 ARKit 就可以更好地对场景进行建模，这样即便设备只是略微晃动，也不会影响追踪质量。但是一旦用户的动作过多、过快或者晃动过于激烈，就会导致图像变得模糊，或者导致视频帧中要追踪的特征之间的距离过大，从而致使追踪质量的降低。ARCamera 类能够提供追踪状态，此外还能提供导致该状态出现的相关原因，您可以在 UI 上展示这些信息，告诉用户如何解决追踪质量低这个问题。

- 给水平面检测预留点时间来生成清晰的结果，一旦您获得所需的结果后，就禁用水平面检测。一开始对水平面进行检测的时候，所检测到的水平面位置和范围很可能不准确。不过随着时间的推移，只要水平面仍然保持在场景当中，那么 ARKit 就能够较为精确地估计水平面的位置和范围。当场景中有一个比较大的平坦表面的话，就算您已经使用过这个水平面来放置内容，那么ARKit 可能还会继续对水平面的锚点位置、范围和变换点进行修正 。

构建基本的 AR 场景

概览

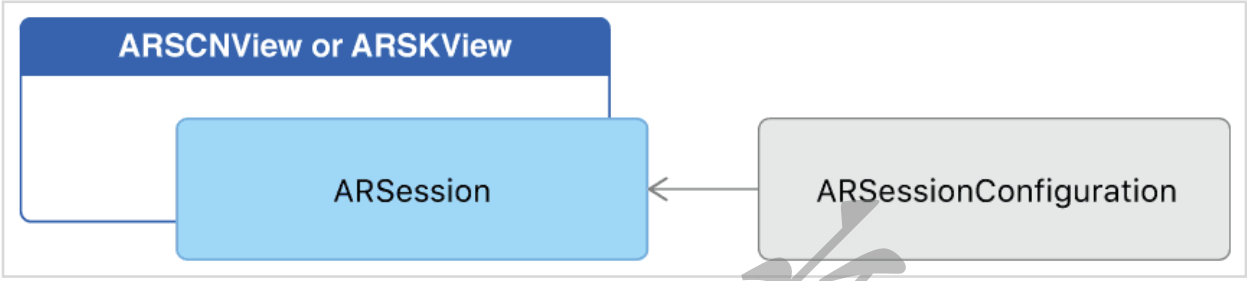
如果您使用了 ARSCNView 或者 ARSKView 类的话，那么 ARKit 就可自行满足创建 AR 场景的基本要求：即每个视图的背景用实时的相机图像来展示，然后还可以渲染您提供的 2D 或者 3D 覆盖物 (overlay)，从而构建出「这些覆盖物实际上是位于现实世界中」这样一种错觉。要使用这些视图类的话，您可以根据您所想要创建的 AR 场景类型来进行配置，然后为覆盖物选定位置和表示方式。

如果您需要构建自定义的视图来展示 AR 场景的话，请参阅「使用 Metal 来展示 AR 场景」一节。

潭州教育

配置 AR Session 并运行

ARSCNView 和 ARSKView 类都是包含在 ARSession 当中的，而 ARSession 对象则用来管理设备动作追踪和进行图像处理的，也就是创建 AR 场景的必需品。但是，要运行 Session，您首先必须选择一种 Session 配置。



您所选择的配置对象的类型，决定了您所创建的 AR 场景的风格和质量：

- 在具备 A9 或者更高版本处理器的 iOS 设备当中，ARWorldTrackingSessionConfiguration 子类提供了高精度的设备动作追踪功能，可以帮助您将虚拟内容「放置」在现实世界中的某个表面上。

- 在 ARKit 所支持的其他设备当中，ARSessionConfiguration 这个基础类则提供了基本的动作追踪功能，可以提供略弱的沉浸式 AR 体验。

要启动 AR Session，那么首先要使用您所需要的配置，来创建 Session 配置对象，然后对 ARSCNView 或者 ARSKView 实例中的 session 对象调用 run(_:options:) 方法：

```
override func viewWillAppear(_ animated: Bool) {
    super.viewWillAppear(animated)

    // 创建 Session 配置
    let configuration = ARWorldTrackingSessionConfiguration()
    configuration.planeDetection = .horizontal

    // 运行视图 Session
    sceneView.session.run(configuration)
}
```

> 重要

漳州教育

只有当视图即将展示在屏幕的时候，才能够运行视图 Session。

当您配置完 AR Session 之后，就可以使用 SceneKit 或者 SpriteKit ，来将虚拟内容放置到视图当中了。

使用 SpriteKit 来提供 2D 虚拟元素

使用 SpriteKit 在您的 3D AR 场景中放置二维图像。

要将 SpriteKit 元素放进增强现实当中，您首先需要运行 AR Session（请参阅「构建基本的 AR 场景」一节）。

SpriteKit 只包含有 2D 元素，但是增强现实则涉及到现实世界的 3D 空间。因此，使用 ARSKView 类来创建 AR 场景，这样就可以在真实世界对应的 3D 位置 (ARAnchor 对象) 中放置 2D 精灵元素 (SKNode) 对象)。当用户移动设备的时候，视图会自动旋转并缩放与锚点相对应的 SpriteKit 结点，看起来这些元素能够追踪相机所看到的真实世界。

举个例子，您可以将 2D 图像以漂浮的方式，放置在 3D 空间当中：

```
// Create a transform with a translation of 0.2 meters in front of the camera.
var translation = matrix_identity_float4x4
translation.columns.3.z = -0.2
let transform = simd_mul(view.session.currentFrame.camera.transform, translation)

// Add a new anchor to the session.
let anchor = ARAnchor(transform: transform)
view.session.add(anchor: anchor)

// (Elsewhere...) Provide a label node to represent the anchor.
func view(_ view: ARSKView, nodeFor anchor: ARAnchor) -> SKNode? {
    return SKLabelNode(text: "👾")
}
```

上面的这个 view(_:nodeFor:) 方法将会返回一个 SKLabelNode 对象，用以展示文本标签。与大多数 SpriteKit 结点一样，这个类将会创建一个 2D 的可视化表示，因此 ARSKView 类将会以广告牌的样式来展示这个结点：精灵可以通过（围绕 z 轴）缩放以及旋转，让其看起来能够跟随锚点的 3D 位置，但是却始终面向相机。

潭州教育

使用 SceneKit 来提供 3D 虚拟元素

> 使用 SceneKit 在您的 AR 场景中放置逼真的三维图像。

要将 SceneKit 元素放到增强现实当中的话，您首先需要运行 AR Session（请参阅「构建基本的 AR 场景」一节）。

由于 ARKit 会自动将 SceneKit 空间与现实世界进行匹配，因此只需要放置一个虚拟对象，然后让其位于一个看起来比较真实的位置，这就需要您适当地设置 SceneKit 对象的位置。例如，在默认配置下，以下代码会将一个 10 立方厘米的立方体放置在相机初始位置的前 20 厘米处：

```
let cubeNode = SCNNode(geometry: SCNBox(width: 0.1, height: 0.1, length: 0.1,
chamferRadius: 0))
cubeNode.position = SCNVector3(0, 0, -0.2) // SceneKit/AR coordinates are in
meters
sceneView.scene.rootNode.addChildNode(cubeNode)
```

上述代码直接在视图的 SceneKit 场景中放置了一个对象。该对象会自动追踪真实世界的位置，因为 ARKit 将 SceneKit 空间与现实世界空间互相匹配了起来。

此外，您也可以使用 ARAnchor 类来追踪现实世界的位置，可以自行创建锚点并将其添加到 Session 当中，也可以对 ARKit 自动创建的锚点进行观察 (observing)。例如，当水平面检测启用的时候，ARKit 会为每个检测到的水平面添加锚点，并保持更新。要为这些锚点添加可视化内容的话，就需要实现 ARSCNViewDelegate 方法，如下所示：

```
func renderer(_ renderer: SCNSceneRenderer, didAdd node: SCNNode, for anchor:
ARAnchor) {
    // This visualization covers only detected planes.
    guard let planeAnchor = anchor as? ARPlaneAnchor else { return }

    // Create a SceneKit plane to visualize the node using its position and
extent.
    let plane = SCNPlane(width: CGFloat(planeAnchor.extent.x), height:
CGFloat(planeAnchor.extent.z))
    let planeNode = SCNNode(geometry: plane)
```

潭州教育

```
    planeNode.position = SCNVector3Make(planeAnchor.center.x, 0,
planeAnchor.center.z)

    // SCNPlanes are vertically oriented in their local coordinate space.
    // Rotate it to match the horizontal orientation of the ARPlaneAnchor.
    planeNode.transform = SCNMatrix4MakeRotation(-Float.pi / 2, 1, 0, 0)

    // ARKit owns the node corresponding to the anchor, so make the plane a child
node.
    node.addChildNode(planeNode)
}
```

遵循设计 3D 资源时的最佳实践

- 使用 SceneKit 基于物理引擎的照明模型，以便获得更为逼真的外观。（参见 SCNMaterial 类和 SceneKit 示例代码项目当中的 「Badger: Advanced Rendering」 。）
- 打上环境光遮蔽阴影 (Bake ambient occlusion shading)，使得物体在各种场景照明条件下能够正常亮起。
- 如果您打算创建一个虚拟对象，并打算将其放置在 AR 的真实平面 (real-world flat surface) 上，那么请在 3D 素材中的对象下方，添加一个带有柔和阴影纹理的透明平面。

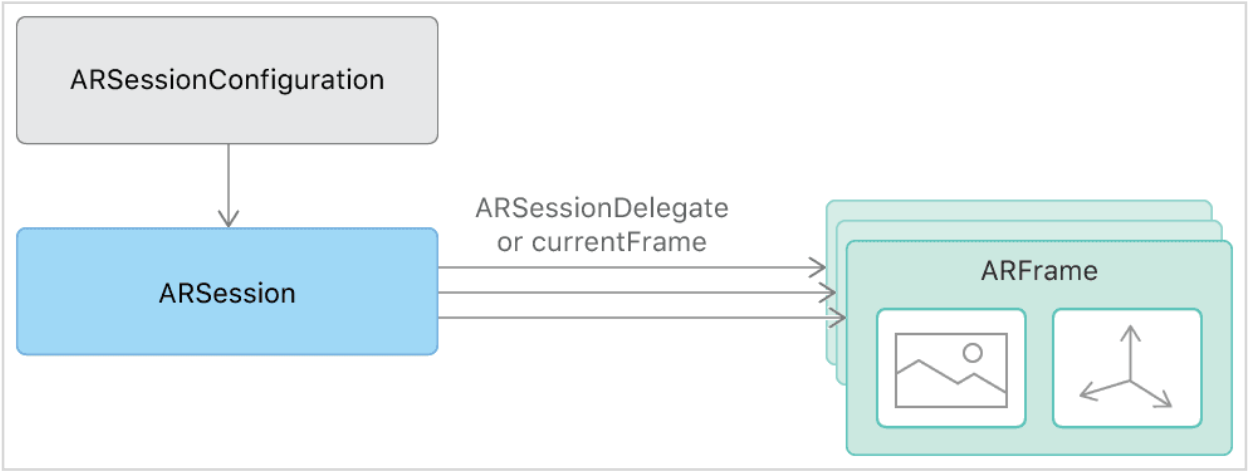
使用 Metal 来展示 AR 场景

> 通过渲染相机图像，以及使用位置追踪信息来展示覆盖物，从而来构建自定义的 AR 视图场景。

概览

ARKit 中内置了一些视图类，从而能够轻松地用 SceneKit 或者 SpriteKit 来展示 AR 场景。然而，如果您使用的是自己的渲染引擎（或者集成了第三方引擎），那么 ARKit 还提供了自定义视图以及其他的支持环境，来展示 AR 场景。

遵州教育



在所有的 AR 场景中，首先就是要配置一个 ARSession 对象，用来管理摄像头拍摄和对设备动作进行处理。Session 定义并维护现实空间和虚拟空间之间的关联关系，其中，现实空间是用户所处的世界，虚拟空间是可对可视化内容进行建模的世界。如果要在自定义视图当中展示您的 AR 场景的话，那么您需要：

1. 从 Session 中检索视频帧和追踪信息；
2. 将这些帧图像作为背景，渲染到自定义视图当中；
3. 使用追踪信息，从而在相机图像上方定位并绘制 AR 内容。

从 Session 中获取视频帧和追踪数据

请自行创建并维护 ARSession 实例，然后根据您所希望提供的 AR 场景类型，使用合适的 Session 配置来运行这个实例。（要实现这点的话，请参阅「构建基本的 AR 场景」。）Session 从摄像机当中捕获视频，然后在建模的 3D 空间中追踪设备的位置和方向，并提供 ARFrame 对象。每个 ARFrame 对象都包含有单独的视频帧 (frame) 图像和被捕获时的设备位置追踪信息。

要访问 AR Session 中生成的 ARFrame 对象的话，有以下两种方法，使用何种方法取决于您应用的设计模式是偏好主动拉取 (pull) 还是被动推送 (push)。

如果您倾向于定时获取视频帧的话（也就是主动拉取设计模式），那么请使用 Session 的 currentFrame 属性，这样就可以在每次重绘视图内容的时候，获取当前的帧图像和追踪信息。ARKit Xcode 模板使用了如下方法：

```
// 在 Renderer 类当中，通过 Renderer.update() 来从 MTKViewDelegate.draw(in:) 中调用
func updateGameState() {
    guard let currentFrame = session.currentFrame else {
```

潭州教育

```
        return
    }

    updateSharedUniforms(frame: currentFrame)
    updateAnchors(frame: currentFrame)
    updateCapturedImageTextures(frame: currentFrame)

    if viewportSizeDidChange {
        viewportSizeDidChange = false

        updateImagePlane(frame: currentFrame)
    }
}
```

相反，如果您的应用设计倾向于使用被动推送模式的话，那么请实现 session(_:didUpdate:) 代理方法，当每个视频帧被捕获之后，Session 就会调用这个代理方法（默认每秒捕获 60 帧）。

获得一个视频帧之后，您就需要绘制相机图像了，然后将 AR 场景中包含的所有覆盖物进行更新和展示。

绘制相机图像

每个 ARFrame 对象的 capturedImage 属性都包含了从设备相机中捕获的像素缓冲区 (pixel buffer)。要将这个图像作为背景绘制到自定义视图当中，您需要从图像内容中构建纹理 (texture)，然后提交使用这些纹理进行 GPU 渲染的命令。

像素缓冲区的内容将被编码为双面 (biplanar) YCbCr 数据格式（也成为 YUV）；要渲染图像的话，您需要将这些像素数据转换为可绘制的 RGB 格式。对于 Metal 渲染而言，最高效的方法便是使用 GPU 着色代码 (shader code) 来执行这个转换了。借助 CVMetalTextureCache API，可以从像素缓冲区中生成两个 Metal 纹理——一个用于决定缓冲区的亮度 (Y)，一个用于决定缓冲区的色度 (CbCr) 面。

```
func updateCapturedImageTextures(frame: ARFrame) {
    // 从所提供的视频帧中，根据其中所捕获的图像，创建两个纹理 (Y and CbCr)
    let pixelBuffer = frame.capturedImage
    if (CVPixelBufferGetPlaneCount(pixelBuffer) < 2) {
        return
```

潍州教育

```
    }
    capturedImageTextureY = createTexture(fromPixelBuffer: pixelBuffer,
pixelFormat:.r8Unorm, planeIndex:0)!
    capturedImageTextureCbCr = createTexture(fromPixelBuffer: pixelBuffer,
pixelFormat:.rg8Unorm, planeIndex:1)!
}

func createTexture(fromPixelBuffer pixelBuffer: CVPixelBuffer, pixelFormat:
MTLPixelFormat, planeIndex: Int) -> MTLTexture? {
    var mtlTexture: MTLTexture? = nil
    let width = CVPixelBufferGetWidthOfPlane(pixelBuffer, planeIndex)
    let height = CVPixelBufferGetHeightOfPlane(pixelBuffer, planeIndex)

    var texture: CVMetalTexture? = nil
    let status = CVMetalTextureCacheCreateTextureFromImage(nil,
capturedImageTextureCache, pixelBuffer, nil, pixelFormat, width, height,
planeIndex, &texture)
    if status == kCVReturnSuccess {
        mtlTexture = CVMetalTextureGetTexture(texture!)
    }

    return mtlTexture
}
```

接下来，使用借助颜色变换矩阵将 YCbCr 转换为 RGB 的函数片段，完成这两个纹理的绘制，我们这里将整个渲染命令进行编码。

```
fragment float4 capturedImageFragmentShader(ImageColorInOut in [[stage_in]],
                                            texture2d<float, access::sample>
capturedImageTextureY [[ texture(kTextureIndexY) ]],
                                            texture2d<float, access::sample>
capturedImageTextureCbCr [[ texture(kTextureIndexCbCr) ]]) {

    constexpr sampler colorSampler(mip_filter::linear,
                                   mag_filter::linear,
                                   min_filter::linear);
```

潭州教育

```
    const float4x4 ycbcrToRGBTransform = float4x4(
        float4(+1.164380f, +1.164380f, +1.164380f, +0.000000f),
        float4(+0.000000f, -0.391762f, +2.017230f, +0.000000f),
        float4(+1.596030f, -0.812968f, +0.000000f, +0.000000f),
        float4(-0.874202f, +0.531668f, -1.085630f, +1.000000f)
    );

    // Sample Y and CbCr textures to get the YCbCr color at the given texture
coordinate
    float4 ycbcr = float4(capturedImageTextureY.sample(colorSampler,
in.texCoord).r,
                          capturedImageTextureCbCr.sample(colorSampler,
in.texCoord).rg, 1.0);

    // Return converted RGB color
    return ycbcrToRGBTransform * ycbcr;
}
```

注意
请使用 displayTransform(withViewportSize:orientation:) 方法来确保整个相机图像完全覆盖了整个视图。关于如何使用这个方法，以及完整的 Metal 管道配置代码，请参阅完整的 Xcode 模板。（请使用 “Augmented Reality” 模板来创建一个新的 iOS 应用，然后在弹出的 Content Technology 菜单当中选择 “Metal”。）

追踪并渲染覆盖内容

AR 场景通常侧重于渲染 3D 覆盖物，使得这些内容似乎是从相机中所看到的真实世界的一部分。为了实现这种效果，我们使用 ARAnchor 类，来对 3D 内容相对于现实世界空间的位置和方向进行建模。锚点提供了变换 (transform) 属性，在渲染的时候可供参考。举个例子，当用户点击屏幕的时候，Xcode 模板会在设备前方大约 20 厘米处，创建一个锚点。

```
func handleTap(gestureRecognize: UITapGestureRecognizer) {
    // Create anchor using the camera's current position
    if let currentFrame = session.currentFrame {

        // Create a transform with a translation of 0.2 meters in front of the
camera
```

潭州教育

```
        var translation = matrix_identity_float4x4
        translation.columns.3.z = -0.2
        let transform = simd_mul(currentFrame.camera.transform, translation)

        // Add a new anchor to the session
        let anchor = ARAnchor(transform: transform)
        session.add(anchor: anchor)
    }
}
```

在您的渲染引擎当中，使用每个 ARAnchor 对象的 transform 属性来放置虚拟内容。Xcode 模板在内部的 handleTap 方法中，使用添加到 Session 当中每个锚点来定位一个简单的立方体网格 (cube mesh)：

```
func updateAnchors(frame: ARFrame) {
    // Update the anchor uniform buffer with transforms of the current frame's anchors
    anchorInstanceCount = min(frame.anchors.count, kMaxAnchorInstanceCount)

    var anchorOffset: Int = 0
    if anchorInstanceCount == kMaxAnchorInstanceCount {
        anchorOffset = max(frame.anchors.count - kMaxAnchorInstanceCount, 0)
    }

    for index in 0..<anchorInstanceCount {
        let anchor = frame.anchors[index + anchorOffset]

        // Flip Z axis to convert geometry from right handed to left handed
        var coordinateSpaceTransform = matrix_identity_float4x4
        coordinateSpaceTransform.columns.2.z = -1.0

        let modelMatrix = simd_mul(anchor.transform, coordinateSpaceTransform)

        let anchorUniforms = anchorUniformBufferAddress.assumingMemoryBound(to: InstanceUniforms.self).advanced(by: index)
        anchorUniforms.pointee.modelMatrix = modelMatrix
    }
```

潭州教育

```
}
```

注意
在更为复杂的 AR 场景中，您可以使用点击测试或者水平面检测，来寻找真实世界当中曲面的位置。要了解关于此内容的详细信息，请参阅 planeDetection 属性和 hitTest(_:types:) 方法。对于这两者而言，ARKit 都会生成 ARAnchor 对象作为结果，因此您仍然需要使用锚点的 transform 属性来放置虚拟内容。

根据实际光照度进行渲染

当您在场景中配置用于绘制 3D 内容的着色器时，请使用每个 ARFrame 对象当中的预计光照度信息，来产生更为逼真的阴影：

```
// in Renderer.updateSharedUniforms(frame:):
// Set up lighting for the scene using the ambient intensity if provided
var ambientIntensity: Float = 1.0
if let lightEstimate = frame.lightEstimate {
    ambientIntensity = Float(lightEstimate.ambientIntensity) / 1000.0
}
let ambientLightColor: vector_float3 = vector3(0.5, 0.5, 0.5)
uniforms.pointee.ambientLightColor = ambientLightColor * ambientIntensity
```

注意
要了解该示例中的全部 Metal 配置，以及所使用的渲染命令，请参见完整的 Xcode 模板。（请使用 “Augmented Reality” 模板来创建一个新的 iOS 应用，然后在弹出的 Content Technology 菜单当中选择 “Metal”。）

漳州教育