免费样品

Unity 着色器圣经。

从初级到高级的着色器的线性解释。使用 Unity 改进您的游戏图形并成为专业的技术艺术家。

（第一版）

法布里奇奥·埃斯平多拉。
游戏开发者和技术美术师。

《Unity 着色器圣经》，版本 0.1.5b。

DDI 2021-A-11866
国际标准书号 979-883-3189-84-9

学分。

作者。
法布里奇奥·埃斯平多拉。

设计。
巴勃罗·耶贝尔.

技术修订。
丹尼尔·桑塔拉.

翻译、语法和拼写。

马丁·克拉克.

关于作者。

Fabrizio Espíndola 是一位智利视频游戏开发人员，专门从事计算机图形学。他的职业生涯大部分时间都致力于开发视觉效果和技术艺术，他的项目包括《星球大战 - 银河防御》、《龙与地下城 - 战争竞技场》、《Timenaus》、《冰雪奇缘 2》和《Nom Noms》。他目前正在与 Jettelly 的团队一起开发一些独立游戏。

1994 年，《Donkey Kong Country》（超级任天堂）问世后，他对电子游戏产生了极大的热情，这款游戏唤起了他对与该技术相关的工具和技术的浓厚兴趣。迄今为止，他已在该行业拥有十多年的经验，这书中包含了他这段时间所学到的部分知识。

告诉他：他或她为他人做出贡献。

> 一个人成为专业人士时

多年前，在大学期间，他的教授 Freddy Gacitúa 曾经

这些话对他的职业发展非常重要，并促使他需要将自己的知识传授给国际独立视频游戏开发者社区。

Jettelly 于 2018 年 3 月 3 日由 Pablo Yeber 和 Fabrizio Espíndola 正式揭牌。他们共同致力于开发不同的项目，其中《Unity Shaders Bible》因其知识性质而成为最重要的项目之一。

内容。

第一章|着色器编程语言简介。

前言。

无论渲染引擎如何，视频游戏开发人员在开始研究着色器时遇到的最大问题之一是网络上缺乏适合初学者的信息。无论读者是独立开发人员还是专注于 AAA 项目，由于开发此类程序所需的技术知识，进入该主题可能会有点复杂。

尽管面临这一挑战，但由于是多平台，Unity 提供了很大的优势，视频游戏代码只需要编写一次，然后就可以导出到不同的设备，包括游戏机和智能手机。同样，一旦开始着色器世界的冒险，代码只需编写一次，然后软件就会负责针对不同平台（OpenGL、Metal、Vulkan、Direct3D、GLES 20、GLES 3x）的编译。

《Unity Shaders Bible》的创建是为了解决在这个世界上刚开始时遇到的大多数问题。您将首先回顾 Cg 和 HLSL 语言中着色器的结构，然后了解其属性、命令、函数和语法。

您是否知道 Unity 中存在三种类型的渲染管道，每种类型都有自己的品质？在整本书中，您将分析它们，验证 Unity 如何处理图形以将视频游戏投影到计算机屏幕上。

I. 我们将在本书中看到的主题。

本书分为三章，按照线性要求对各要点进行阐述。本书中看到的所有代码都已使用 Visual Studio Code 编辑器进行了测试，并在 Unity 中检查了不同类型的渲染管道。

第一章：着色器编程语言简介。

本章主要介绍开始之前所需的基础知识，例如ShaderLab语言中的着色器结构、属性和连接变量之间的类比、SubShader和命令（ColorMask、Stencil、Blending等）、Cg和HLSL语言的Passes和结构、函数结构、顶点输入分析、顶点输出分析、语义与基元之间的类比、顶点着色器阶段结构、片段着色器阶段结构、矩阵等等。本章是了解有关着色器在 Unity 中如何工作的基本概念的起点。

第二章：光照、阴影和表面。

本节讨论高度相关的问题，例如：法线贴图及其实现、反射贴图、照明和阴影分析、基本照明模型、表面分析、数学函数、镜面反射和环境光。此外，还回顾了 Shader Graph 及其结构、HLSL 函数、节点、属性等。在本章中，您将使用简单的照明概念使您的视频游戏看起来很专业。

第三章：计算着色器、光线追踪和球体追踪。

在这里，您将实践高级概念，例如：计算着色器的结构、缓冲区变量、内核、球体跟踪实现、隐式曲面、形状和算法、光线跟踪简介、配置和高质量渲染。本章的学习将通过调查 GPGPU 编程（一般用途 GPU），使用 .compute 类型着色器，尝试球体追踪技术并在 HDRP 中使用直接光线追踪 (DXT)。

二.建议。

使用代码编辑器非常重要智能感知图形语言编程，特别是 Cg 或 HLSL。团结有视觉工作室代码，这是 Visual Studio Community 的更紧凑版本。此编辑器包含一些将 IntelliSense 添加到 C#、ShaderLab 和 HLSL 的扩展。

对于使用 Visual Studio Code 编辑器的用户，建议安装以下扩展：用于 **Visual Studio** 代码的 **C#**（微软），**VS Code** 的着色器语言支持（奴隶），**ShaderLab VS** 代码（阿姆洛维），**Unity** 代码片段（克莱伯·席尔瓦）。

三. 这本书是为谁而写的。

本书是为希望提高图形技能或创建专业效果的 Unity 开发人员编写的。假设读者已经了解、理解并能够访问 Unity 界面；因此，不再详细讨论。

具备 C# 或 C++ 的先前知识对于理解本书中介绍的内容将是一笔巨大的财富；然而，这并不是唯一的要求。

拥有一些算术、代数和三角学的基础知识对于理解更高级的概念至关重要。尽管如此，我们仍将审查数学运算和函数，以充分理解您正在开发的内容。

四. 词汇表。

鉴于其性质，本书中会有一些与其他内容不同的短语和单词，它们很容易识别，因为它们被突出显示以强调解释或概念。同样，HLSL 中也有代码块来说明一些功能。

有些单词以粗体显示，这也用于强调代码行。其他技术定义以大写字母开头（例如 Vertex），而那些具有恒定性质的定义将完全以相同的样式呈现（例如 RGBA）。

在函数定义中，参数用首字母缩略词 RG 显示（例如，N_{RG}）和空间用 AX 显示的坐标（例如，$Y_{斧头}$）。此外，还有包含三个句点 (...) 的代码块，这些代码块指的是代码中默认包含的变量或函数。

五、勘误表。

在撰写本书时，我们采取了一切预防措施来确保其内容的真实性。即便如此，请记住，我们是人类，很可能有些观点没有解释清楚，或者在拼写/语法纠正时出现错误。

如果您发现概念错误、代码或其他错误，请通过电子邮件通知我们，我们将不胜感激在**contact@jettelly.com**表明**USB**　勘误表在学科领域；这样，你就会帮助其他读者通过在以下版本中进行改进来减少他们的挫败感。

此外，如果您觉得本书缺少一些有趣的部分，欢迎您给我们发送电子邮件，我们将在未来的版本中包含这些信息。

六. 资产和捐赠。

本书拥有独家资源来强化下载中包含的内容。这些是使用 Unity 2020.3.21f1 开发的，并在内置和可编写脚本中进行了分析渲染管线。 **learn.jettelly.com/usb-resources**

您在 Jettelly 中看到的所有作品都是由其自己的成员开发的，其中包括绘图、设计、视频、音频、教程以及您在该品牌中看到的所有内容。

Jettelly 是一家独立视频游戏开发工作室，您的支持对我们至关重要。如果您想在经济上支持我们，您可以直接通过我们的 PayPal 帐户进行操作。 **paypal.com/paypalme/jettelly**

七. 海盗行为。

在未经我们同意的情况下复制、复制或提供本材料之前，请记住：Jettelly 是一家独立、自筹资金的工作室。任何非法行为都可能影响我们的诚信作为一个开发团队。

本书已获得版权专利，我们将认真保护我们的许可。如果您在 Jettelly 以外的平台上发现本书或发现非法副本，请通过以下方式联系我们：**contact@jettelly.com**（如有必要，附上链接）。这样，我们就可以找到解决办法了。先感谢您。

第一章

着色器编程语言简介。

初步观察。

几年前，当我刚开始学习 Unity 着色器时，由于各种原因，我在书中找到的大部分内容都很难理解。我还记得那

天我试图理解语义函数 POSITION[n]。然而，当我最终设法找到它时

定义中，我发现了以下语句：

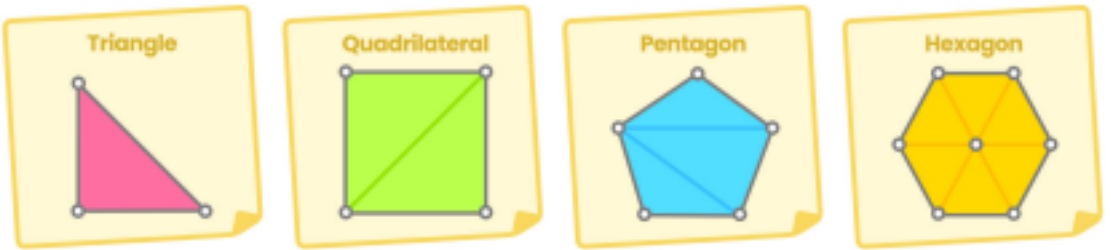
对象空间中的顶点位置。

那一刻，我问自己，顶点在对象空间中的位置是什么？然后我明白，在开始阅读这个主题之前，我必须先了解一些先前的信息。根据我的经验，我已经能够确定至少四个有助于理解着色器及其结构的基本领域，例如：

多边形对象的属性。
渲染管线的结构。
矩阵和坐标系。

1.0.1。多边形对象的属性。

多边形一词源自希腊语 πολύγωνος (polúgōnos) 由poly（许多）和gnow（角）组成。根据定义，多边形是指由线段包围的封闭平面图形。



多边形对象的属性●●●（图*1.0.1a*）

A原始是由多边形构成的三维几何对象，在不同的开发软件中作为预定义对象。在 Unity、Maya 或 Blender 中，您可以找到其他基元。最常见的是：

球体。
盒子。
四边形。
气缸。
胶囊。

所有这些物体的形状不同，但具有相似的性质；都有：

顶点。
切线。
法线。
紫外线坐标。
颜色。

它们存储在称为的数据类型中网。

所有这些属性都可以在着色器中独立访问并保存在向量中。这非常有用，因为您可以修改它们的值，从而产生有趣的效果。为了更好地理解这个概念，下面是多边形对象属性的简单定义。

多边形对象的属性●●●

（图*1.0.1b*）

1.0.2。顶点。

对象的顶点对应于定义二维或三维空间中的表面面积的点集。在 Maya 和 Blender 中，顶点表示为对象网格的交点，类似于一组原子（分子）。

这些要点有两个主要特征：

它们是 Transform 组件的子组件。

它们根据物体总体积的中心有一个确定的位置。

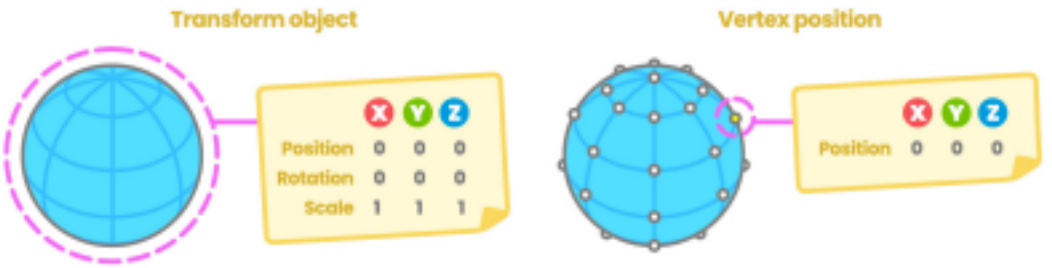
这是什么意思？ Maya 3D 有两个与对象关联的默认节点。这些被称为转换和形状。

与 Unity 中一样，Transform 节点定义对象相对于对象枢轴的位置、旋转和缩放。Shape 节点是 Transform 节点的子节点，包含几何属性，即对象顶点相对于其体积的位置。

这意味着对象的顶点集可以移动、旋转或缩放，同时特定顶点的位置也可以改变。

HLSL 中的 POSITION[n] 语义专门允许访问顶点相对于其体积的位置，即形状节点从 Maya 导出的配置。

顶点●●●



（图*1.0.2a*）

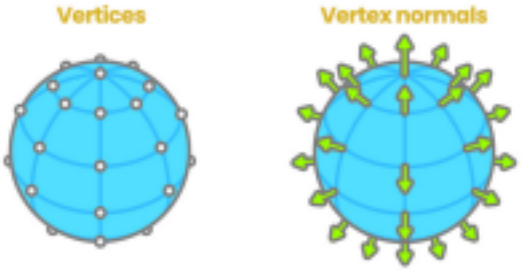
1.0.3。法线。

想象一下，一位朋友被要求在一张白纸的正面画画。如果一张白纸两面都相等，如何确定哪一张是正面？这就是为什么法线存在。

A普通的对应于多边形表面上的垂直矢量，用于确定面或顶点的方向或方位。

在 Maya 中，通过选择属性“顶点法线”来可视化对象法线。这使我们能够看到顶点在空间中的指向位置，并确定物体不同面之间的硬度级别。

法线●●●



（图 *1.0.3a.* 每个顶点法线的图形表示）

1.0.4。切线。

表面沿水平纹理方向。

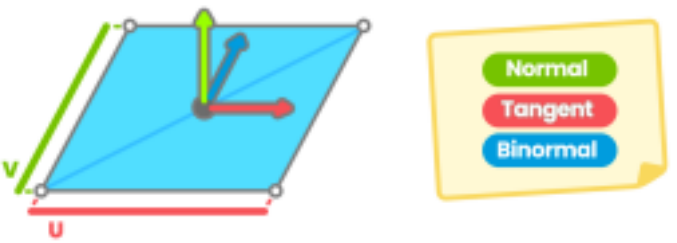
“

切线是沿着网格的长度单位的向量

“

根据Unity官方文档：

这是什么意思？查看图 1.0.4a 以了解其本质。这切线是一个标准化向量，遵循每个几何面上 UV 的 U 坐标方向。它的主要功能是生成一个称为Tangent-Space的空间。



切线●●●（图 *1.0.4a*。默认情况下，无法在着色器中访问副法线。相反，需要根据法线和切线来计算它们 ）

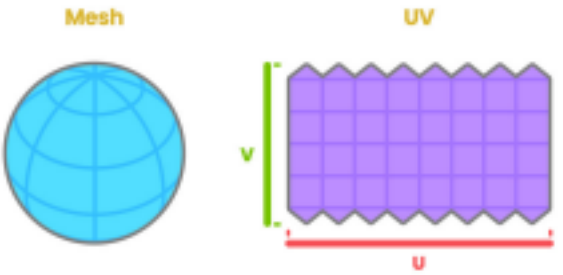
稍后将在第二章第 6.0.1 节以及双法线用于在对象上实现法线贴图。

1.0.5。紫外线坐标。

每个人都把自己最喜欢的角色的皮肤换成了更好的皮肤。 UV 坐标与此概念直接相关，因为它们允许您将二维纹理定位在三维对象的表面上。这些坐标充当参考点，控制纹理映射中到网格中每个顶点的相应纹理像素。

在 UV 坐标上定位顶点的过程称为紫外映射UV 是一个创建、编辑和组织 UV（显示为对象网格的扁平二维表示）的过程。您可以在着色器中访问此属性，以在 3D 模型上定位纹理或在其中保存信息。

紫外坐标●●●



（图 *1.0.5a*。*UV* 贴图中顶点可以以不同的方式排列）

19 号

UV 坐标的面积等于 0.0f 到 1.0f 之间的范围，其中第一个对应于起点，第二个对应于终点。

紫外坐标●●●

（图 *1.0.5b.* 笛卡尔平面中 *UV* 坐标的图形参考）

1.0.6。顶点颜色。

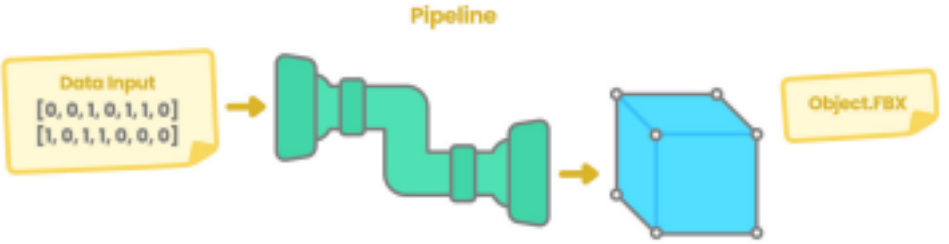
当您从 3D 软件导出对象时，它会通过照明或与另一种颜色相乘来为受影响的对象分配一种颜色。这种颜色被称为顶点颜色默认为白色，RGBA 通道值为 1.0f。

1.0.7。渲染管线架构。

在当前版本的 Unity 中，渲染管线分为三种类型：内置RP，普遍的RP（称为轻的在以前的版本中），以及高清RP。

值得问的是，什么是渲染管线？要回答这个问题，首先要了解管道的概念。

管道是执行更大任务操作的一系列阶段。那么，渲染管线指的是什么？这个概念可以被认为是多边形对象渲染到我们的计算机屏幕上必须采取的完整过程；它就像一个物体穿过超级马里奥管道直到到达最终目的地。



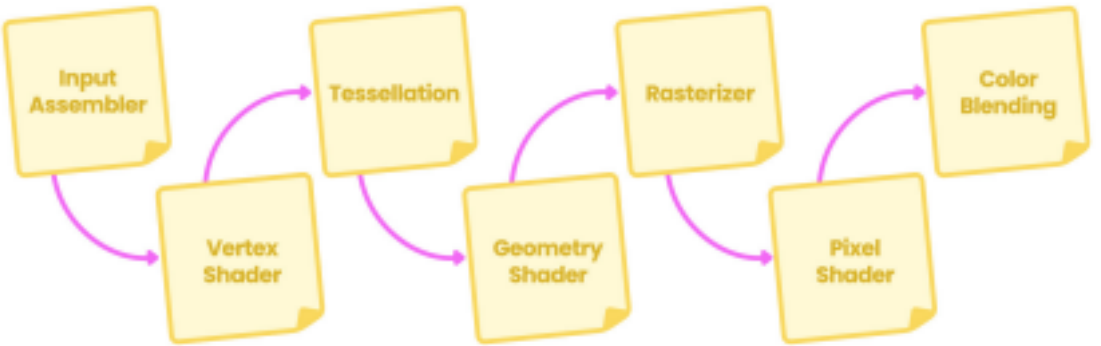
渲染管线架构●●●（图*1.0.7a*）

因此，每个渲染管道都有自己的特性，并且根据您使用的类型，将影响屏幕上对象的外观和优化：材质属性、光源、纹理以及内部发生的所有功能在着色器内。

现在，这个过程是如何发生的？为此，您必须了解其基本架构。Unity 将该架构分为四个阶段，分别是：

申请阶段。
几何处理阶段。
光栅化阶段。
像素处理阶段。

请注意，这对应于实时渲染引擎的渲染管道的基本模型。上述每个阶段都有现在将要定义的线程。



渲染管线架构●●●（图 *1.0.7b.* 逻辑渲染管线 ）

1.0.8。申请阶段。

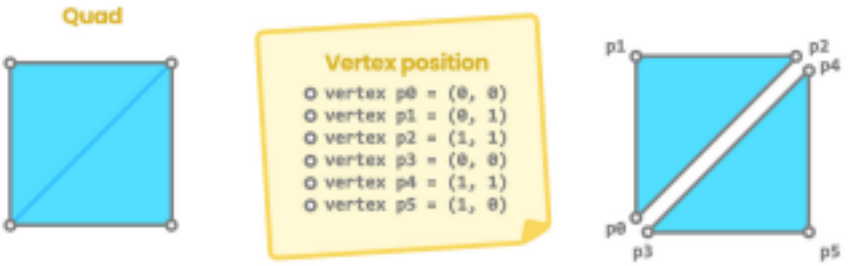
应用程序阶段从 CPU 开始，负责场景中发生的各种操作，例如：

碰撞检测。
纹理动画。
键盘输入。

鼠标输入等等。

它的功能是读取存储的内存数据以稍后生成图元(例如，三角形、线、顶点)。在应用阶段结束时，所有这些信息都被发送到几何处理阶段，然后通过矩阵乘法生成顶点变换。

申请阶段●●●



（图*1.0.8a.*四边形中顶点的局部位置）

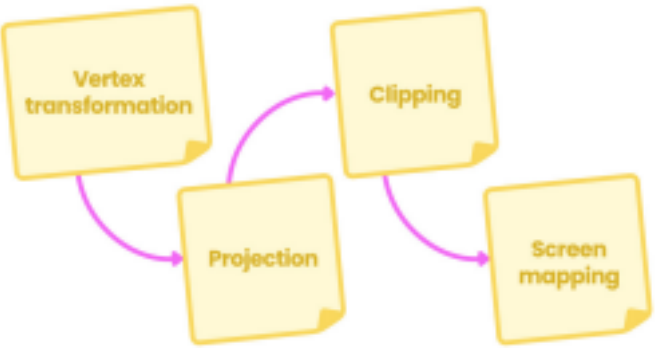
1.0.9。几何处理阶段。

您在计算机屏幕上看到的图像是 GPU 向 CPU 请求的。这些请求分两个主要步骤执行：

渲染状态的配置，对应于从几何处理到像素处理的阶段集。
然后，对象被绘制在屏幕上。

几何处理阶段发生在 GPU 中，负责对象的顶点处理，该阶段分为四个子进程，分别是：

顶点着色。
投影。
剪裁。
屏幕映射。



几何处理阶段●●●（图*1.0.9a*）

一旦在应用程序阶段组装了基元，顶点着色（更广为人知的名称）顶点着色器阶段，执行两项主要任务：

它计算对象顶点位置。
然后它将其位置转换为不同的空间坐标，以便将它们投影到计算机屏幕上。

此外，在此子流程中，您可以选择要传递到以下阶段的属性，例如法线、切线、UV 坐标等。

在此过程中会发生投影和剪切，这根据场景中的相机属性而变化：也就是说，如果配置为透视或正交（平行）。值得一提的是，完整的渲染过程仅发生在相机视锥体（也称为视图空间 ）内的那些元素。

为了理解这个过程，假设场景中有一个球体，其中一半位于相机的平截头体之外。仅投影位于截锥体内的球体区域

随后在屏幕上进行剪裁（clipping），而球体中看不见的区域将在渲染过程中被丢弃。

几何处理阶段●●●

（图*1.0.9b*）

一旦剪切的对象进入内存，它们就会被发送到屏幕映射。在此阶段，场景中的三维对象被转换为2D屏幕坐标，也称为屏幕或窗口坐标。

1.1.0。光栅化阶段。

第三阶段对应于光栅化。此时，对象具有 2D 屏幕坐标，因此必须找到投影区域中的像素。查找屏幕上对象周围所有像素的过程称为光栅化。这个过程可以看作是场景中的对象和屏幕上的像素之间的同步点。

对于每个对象，光栅化器执行两个过程：

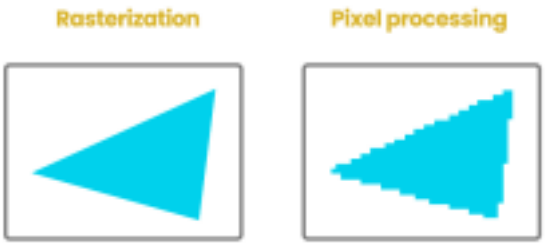
三角形设置。
三角形遍历。

三角形设置生成将发送到的数据三角形遍历。它包括屏幕上对象边缘的方程。之后，三角形遍历列出了多边形对象区域覆盖的像素。这样就生成了一组像素，称为片段；从此这个词片段着色器，也用于指代单个像素。

1.1.1.像素处理阶段。

使用先前过程中的插值，当所有像素准备好投影到屏幕上时，最后阶段开始。此时，片段着色器阶段，也称为像素着色器阶段，开始并负责每个像素的可见性。它的作用是处理像素的最终颜色，然后将其发送到颜色缓冲器。

像素处理阶段●●●



（图*1.1.1a*。几何体覆盖的区域转换为屏幕上的像素）

1.1.2.渲染管线的类型。

如您所知，Unity 中的渲染管线分为三种类型。默认情况下，有内置相反，RP 对应于属于该软件的最旧的引擎，普遍的RP 和高清RP 属于一种称为 Render Pipeline 的类型可编写脚本RP，哪个更多是最新的并且已经过预先优化以获得更好的图形性能。

渲染管线的类型●●●

（图*1.1.2a*。当您在*Unity*中创建新项目时，您可以在这三种渲染引擎之间进行选择。您的选择取决于手头项目的需求）

不管渲染管道如何，如果你想在屏幕上生成图像，你就必须经过管道。

管道可以有不同的处理路径，称为渲染路径;就好像 1.0.7 节中的示例管道有不止一种方法可以到达目的地。

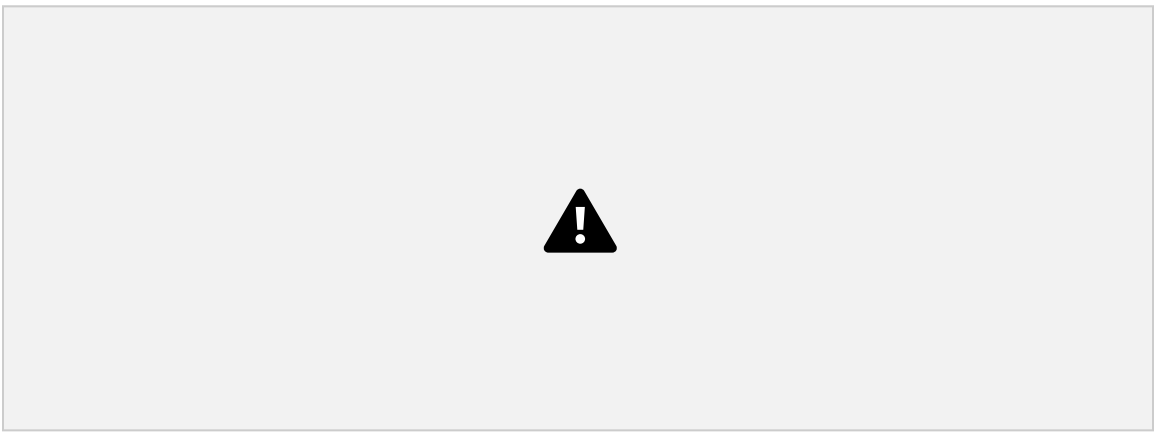
渲染路径对应于与照明和着色对象相关的一系列操作。这允许您以图形方式处理照明场景（例如，具有定向光和球体的场景）。

其中您可以找到：

前向渲染。
延迟着色。
遗产被推迟。
旧版顶点点亮。

其中每一个都有不同的功能和性能特征。

在Unity中，默认的渲染路径对应于前向渲染;这是 Unity 中包含的三种渲染管线（内置、通用和高清）的初始路径。这是因为它具有更大的显卡兼容性和光照计算限制，使其成为更优化的过程。



渲染管线的类型●●●（图*1.1.2b*。要在内置渲染管道中选择渲染路径，您必须进入层次结构，选择主相机，并在属性渲染路径中您可以根据项目的需要更改配置）

要理解这个概念，请想象场景中的一个物体和直射光。它们之间的相互作用基于以下两个基本概念：

照明特性。
物体材料特性。

这种相互作用称为照明模型。
基本光照模型对应于三个不同属性的总和：

环境颜色。

漫反射。

镜面反射。

光照计算在着色器内进行，可以通过顶点或片段来完成。当通过顶点计算光照时，称为每顶点照明并在顶点着色器阶段执行。同样，当按片段计算时，称为每个片段或者每像素照明并在片段着色器阶段执行。

1.1.3.前向渲染。

向前是默认的渲染路径，支持材质的所有典型功能，包括法线贴图、单个像素照明、阴影等。该渲染路径有两个不同的代码编写通道，您可以在着色器中使用它们，第一个，基础通道和第二个额外通行证。

在基本通道中您可以定义**ForwardBase** 灯光模式在附加通道中，您可以定义前进添加灯光模式用于额外的照明计算。两者都是带有光照计算的着色器的特征函数。基础通道可以按像素处理定向光，并且如果场景中存在多个定向光，则会优先考虑最亮的光。此外，基础通道还可以处理Light Probes、全局照明和环境照明（天空之光）。

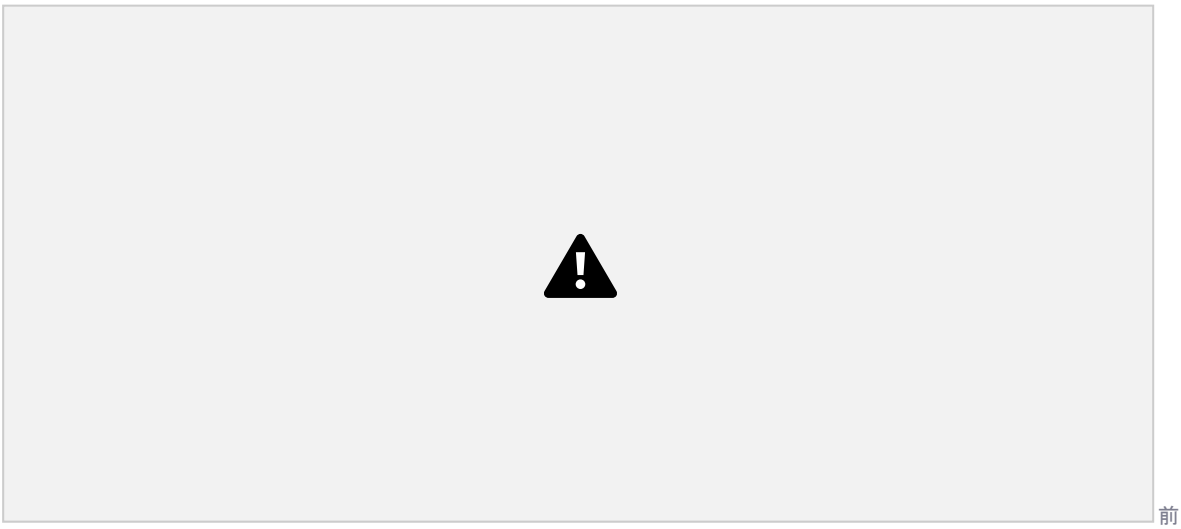
顾名思义，附加通道可以处理附加灯光（点光源、聚光灯、区域光）或影响对象的阴影。这是什么意思？如果场景中有两盏灯，您的对象将仅受其中一盏灯的影响。但是，如果您为此配置定义了附加通道，那么它将受到两者的影响。

需要考虑的一点是每个照明通道都会生成一个独立的抽奖。这是什么意思？根据定义，绘制调用是每次在计算机屏幕上绘制元素时在 GPU 中进行的调用图形。这些调用是需要大量计算的过程，因此需要将它们保持在尽可能低的水平，如果您正在处理移动设备的项目，则更是如此。

为了理解这个概念，假设场景中有四个球体和一个定向光。每个球体本质上都会生成对 GPU 的调用，这意味着它们每个球体都会默认生成一个独立的绘制调用。

同样，定向光会影响场景中发现的所有球体，因此，它将为每个球体生成一个额外的绘制调用。

这主要是因为着色器中包含了第二遍来计算阴影因此，四个球体加上单向光总共会产生八个图形调用。



（图*1.1.3a*。上图中可以看到有光源时*Draw Calls*的增加。计算包括环境颜色和光源作为对象）

确定了基本通道后，如果在着色器中添加另一个通道，则将为每个对象添加另一个绘制调用，因此，图形负载将相应增加。

1.1.4.延迟着色。

此渲染路径确保只有一个照明通道计算场景中的每个光源，并且仅在受光源影响的像素中进行计算，所有这一切都是通过几何体和照明的分离实现的。这是一个优势，因为可以生成大量影响不同对象的光，从而提高最终渲染的保真度，但名义上会增加 GPU 上的每像素计算。

虽然 Deferred Shading 在计算多个光源时优于 Forward，但它也有一些限制，例如根据 Unity 官方文档，Deferred Shading 需要具有多个 Render Targets 的显卡、Shader Model 3.0 或更高版本，并且支持用于深度渲染纹理。

在移动设备上，此配置仅适用于至少支持 OpenGL ES 3.0 的设备。

关于此渲染路径的另一个重要考虑因素是它只能在具有透视相机的项目中使用。延迟着色不支持正交投影。

1.1.5。我应该使用什么渲染管道？

过去只有内置RP，因此启动2D或3D项目非常容易。然而，现在项目必须根据需求来启动，那么你可能会想，项目需要什么？要回答这个问题，必须考虑以下因素：

如果正在为 PC 开发视频游戏，您可以使用三个可用的 Unity 渲染管道中的任何一个，因为通常 PC 的计算能力比移动设备或控制台更强。那么，如果视频游戏针对的是高端设备，它是否需要在图形上看起来很逼真？如果是这样，您可以从高清和内置 RP 开始。

如果视频游戏需要中等清晰度的图形，您可以使用通用 RP，或者像前面的情况一样，也可以使用内置 RP。现在，为什么内置 RP 在这两种情况下都作为一个选项出现？

与之前的渲染管线不同，此渲染管线更加灵活，因此技术性更强，并且没有预先优化。高清 RP 已预先优化为生成高端图形，Universal RP 已针对中端图形进行了预先优化。

选择渲染管道时的另一个重要因素是着色器。一般来说，在 High-Definition 和 Universal RP 中，着色器都是在着色器图，这是一个包，带来了允许通过节点开发着色器的接口。

这带来了积极和消极的一面。一方面，您可以通过节点可视化地生成着色器，而无需在 HLSL 中编写代码。但是，如果在生产期间（例如从2019年到2022年）想要将unity版本升级到更高版本，则着色器很可能会停止编译，因为Shader Graph具有独立的版本和更新。

在 Unity 中生成着色器的最佳方法是通过 HLSL，因为这种方法可以确保您的程序在不同的渲染管道中进行编译，并且无论 Unity 更新如何，都可以继续工作。稍后在详细回顾 HLSL 中的程序结构时将讨论这个概念。

1.1.6。矩阵和坐标系。

创建着色器时经常看到的概念之一是矩阵。矩阵是遵循一定算术规则的数字元素列表，在计算机图形学中经常使用。

在 Unity 中，矩阵代表空间变换，其中包括：

- UNITY_MATRIX_MVP。
- UNITY_MATRIX_MV。
- UNITY_MATRIX_V。
- UNITY_MATRIX_P。
- UNITY_MATRIX_VP。
- UNITY_MATRIX_T_MV。
- UNITY_MATRIX_IT_MV。
- unity_ObjectToWorld。
- unity_WorldToObject。

所有这些都对应于四乘四矩阵（4x4），即每个矩阵都有四行四列数值。

它们的概念表示如下：

```
UNITY_MATRIX
(
        xx, Yx, Zx, Tx,
        XY,是, Zy, Ty,
        Xz、Yz、Z Z, 兹,
        Xt、Yt、Zt、特瓦
);
```

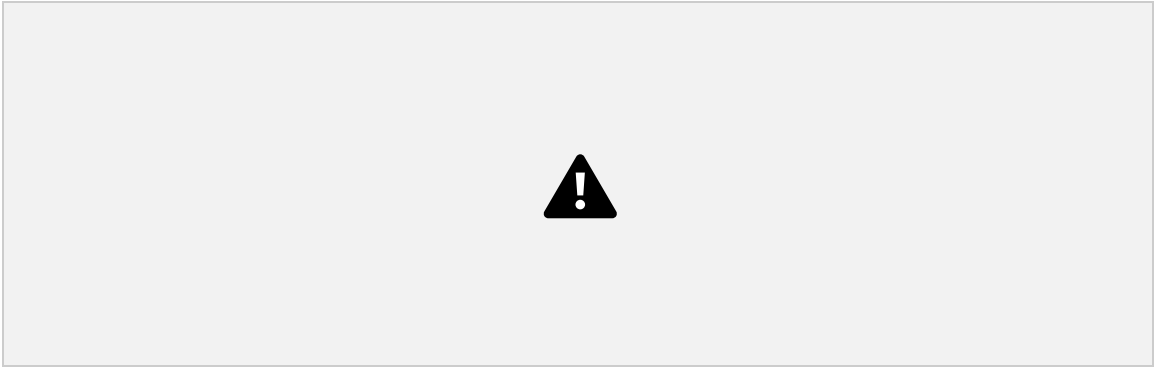
正如之前在 1.0.2 节讨论顶点时所解释的，多边形对象默认有两个节点。在 Maya 中，这些节点称为“变换”和“形状”，两者都负责计算称为“对象空间”的空间中的顶点位置，该空间定义了顶点相对于对象中心位置的位置。

对象中每个顶点的最终值乘以一个称为模型矩阵(UNITY_MATRIX_M)，允许您修改其变换、旋转和缩放值。每次旋转对象、更改其位置或比例时，模型矩阵都会更新。

这个过程是如何发生的？要理解它，请尝试变换场景中的立方体。首先选取位于位置 0.5 的立方体顶点$_{X}$,-0.5$_{和}$,-0.5$_{和}$, 1.0$_{在}$ 相对于它的中心。

值得一提的是，通道W对应的坐标系称为同质，它允许统一处理向量和点。在矩阵变换中，W 坐标可以等于 0 或 1。当n$_{在}$ 等于1时，表示空间中的一个点，等于0时，表示一个方向。

随后，本书将讨论向量与矩阵相乘的系统，反之亦然。



（图*1.1.6a.*单位矩阵指的是矩阵默认值 ）

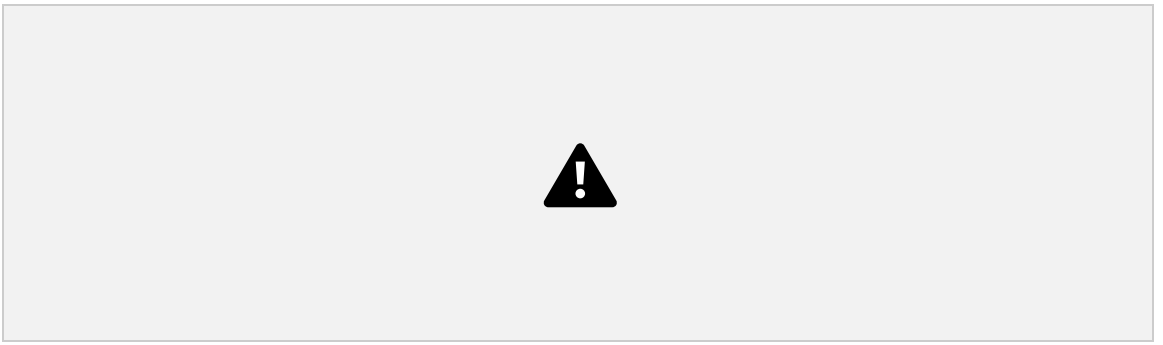
关于矩阵需要考虑的一件事是，只有当第一个矩阵的列数等于第二个矩阵的行数时才能执行乘法。众所周知，该模型矩阵的维度为四行四列，顶点位置的维度为四行一列。

由于模型矩阵中的列数等于顶点位置的行数，因此可以将它们相乘，结果将等于一个四行一列的新矩阵，这将定义一个新位置对于顶点。这个乘法过程发生在对象的所有顶点上，并在顶点着色器阶段。

到目前为止，您已经知道对象空间是指顶点根据自身中心的位置，那么世界空间、视图空间或剪辑空间是什么意思呢？概念基本相同。

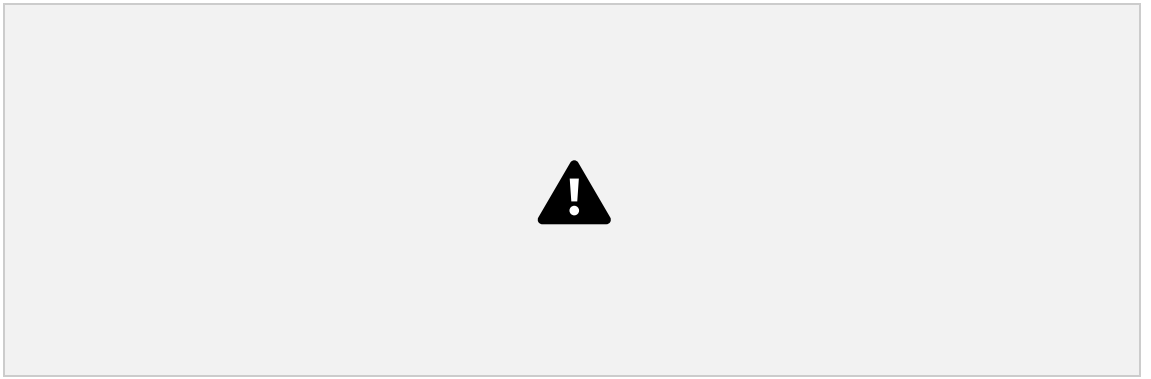
World-Space对应于根据世界中心的顶点的位置；到场景中网格起点之间的距离（0_{X}, $0_{和}$, $0_{和}$, $1_{在}$）以及对象上顶点的位置。

如果要将空间坐标从对象空间转换为世界空间，可以使用内部变量 unity_ObjectToWorld。



（图*1.1.6b*）

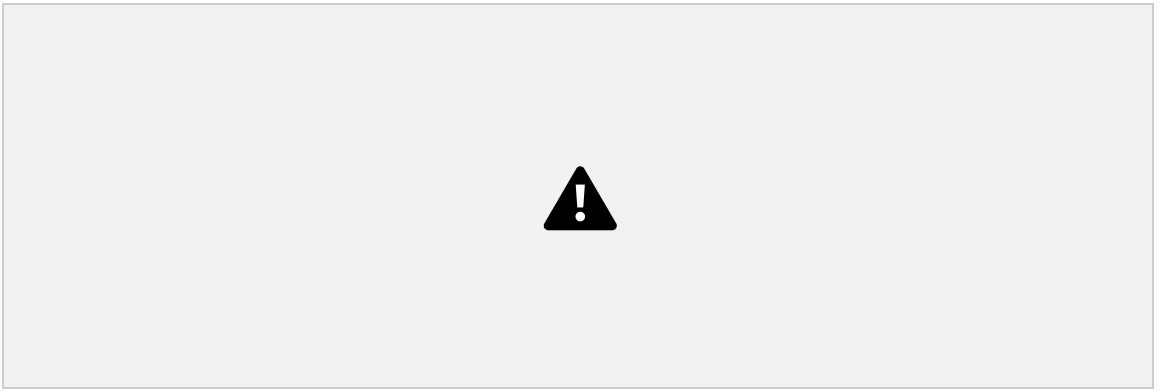
视图空间是指对象的顶点相对于相机视图的位置。如果要将空间坐标从世界空间转换为视图空间，可以使用 UNITY_ MATRIX_V 矩阵。



（图*1.1.6c*）

最后，剪辑空间，也称为投影空间，是指对象顶点相对于相机平截头体的位置。所以这个因素会受到以下因素的影响近剪裁平面、远剪裁平面和视野。

如果要将空间坐标从视图空间转换为剪辑空间，可以使用 UNITY_MATRIX_P 矩阵来完成。



时间和动画●●●（图 *4.2.0b.* 四边形上 *U* 和 *V* 方向的偏移旋转。平铺等于 *4*）

第二章

照明、阴影和表面。

所得向量的大小将与 sin 的函数相关。考虑到上面提到的向量 A 和 B，您可以根据两个向量之间的行列式矩阵计算叉积。

向量 C = X [($A_{和}$ * $B_{和}$）-（$A_{和}$ * $B_{和}$）] 已经$_{和}$ * B_X）-（A_X * $B_{和}$）] Z [（A_X * $B_{和}$）-（$A_{和}$ * B_X）] 通过替换这些值，你就得到了。

向量 C = X [(0 * 0) - (0 * 1)] Y [(0 * 1) - (1 * 0)] Z [(1 * 1) - (0 * 0)]
向量 C = X [(0 - 0)] Y [(0 - 0)] Z [(1 - 0)]
向量 C = (0, 0, 1)

总之，所得向量垂直于其参数。

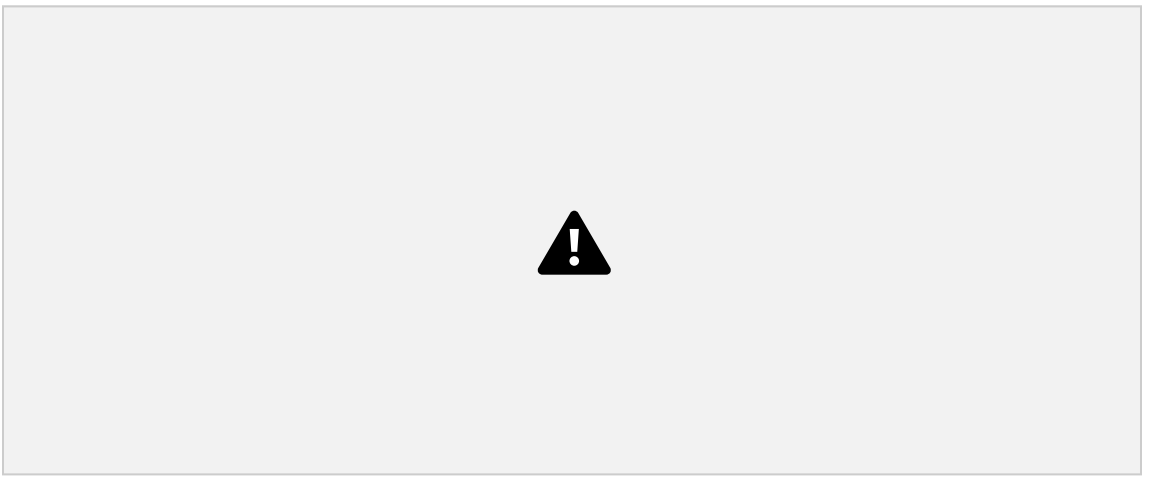
稍后，您将使用十字（A_{RG},B_{RG}）函数来计算法线贴图中副法线的值。


```
固定4碎片（v2f我）：SV_目标
{
    固定4法线贴图=tex2D(_NormalMap, i.uv_normal);
    固定3正常压缩=DXT压缩（法线贴图）；
    浮动3x3TBN_矩阵= 浮点数3x3
    （
        i.tangent_world.xyz,
        i.binormal_world,
        i.正常世界
    ）；
    固定4正常颜色=正常化（我有（正常压缩，TBN_矩阵））；返回固定4（正常颜色，1）；
}
```

上面的例子创建了一个三维向量，称为正常颜色。该向量具有法线贴图和 TBN 矩阵的结果。最后，您以 RGB 形式返回法线的颜色，并将值“one”分配给 A 通道。

值得一提的是，当您将法线贴图导入到 Unity 时，默认情况下，它是在纹理类型默认在你的项目中。

在将法线贴图分配给材质之前，您必须选择纹理，转到检查器，并将其设置为纹理类型法线贴图，否则程序可能无法正常运行。



总碱值矩阵●●●（图*6.0.3a*）

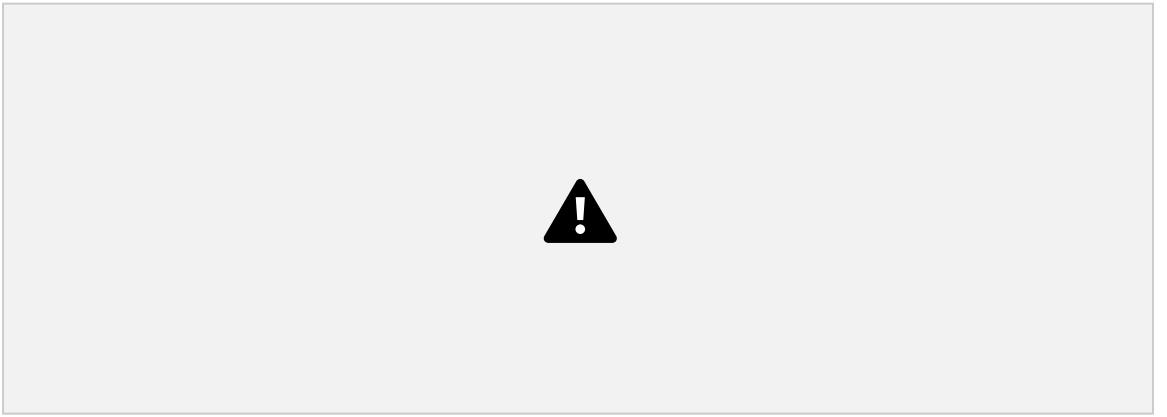
207

镜面反射●●●

```
着色器“USB/USB_镜面反射”
{
    特性{ … }
    子着色器
    {
        标签
        {
            “渲染类型”=“不透明”
            “灯光模式”=《前进基地》
        }
    }
}
```

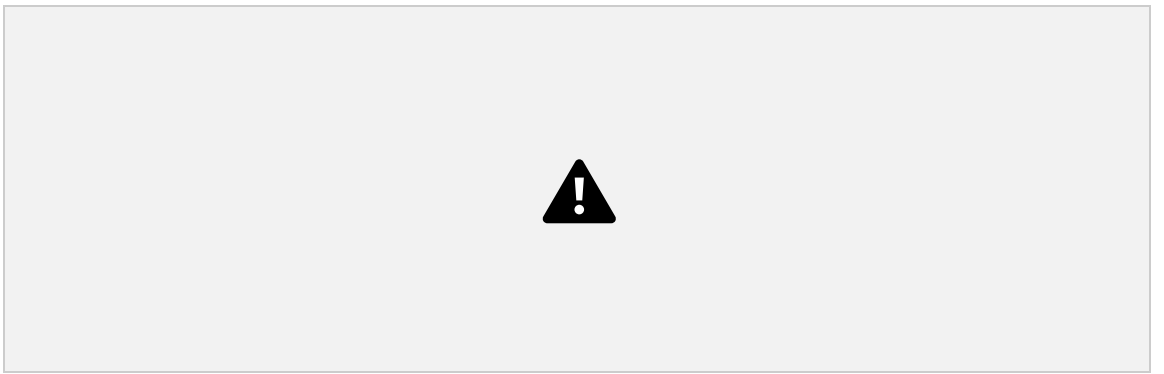
7.0.5。环境反思。

环境反射的发生方式与镜面反射类似。它的区别在于影响表面的光线数量，例如，在一般场景中，镜面反射仅由主光源生成，而环境反射是由每条光线照射到表面（包括从所有角度反射）生成的。



环境反思●●●(图*7.0.5a*)

鉴于其性质，实时计算这种类型的反射会消耗大量 GPU 能力，相反，您可以使用立方体贴图类型纹理。在第一章第 3.0.6 节中，立方体提到的属性正是指这种类型的纹理。



(图*8.0.1b*)

阴影贴图是一种纹理；因此它具有 UV 坐标，并分两个阶段计算：首先，根据光源视点渲染场景。在此过程中，从Z-Buffer中提取深度信息，然后将其作为纹理保存在内存中。在第二种情况下，根据相机视点以通常的方式在 GPU 上绘制场景。您必须在此处计算保存在内存中的纹理的 UV 坐标，以生成阴影并将其应用到您正在使用的对象上。

8.0.2。暗影施法者。

从生成阴影开始。为此，创建一个新的 Unlit 着色器并将其命名为**USB_shadow_map**。在此过程中，您需要两次通过：

一种投射阴影(Shadow Caster)。
另一个接收它们(阴影图)。

因此，您必须做的第一件事是包含第二个通道，它将负责阴影投影。

暗影施法者●●●着色器“USB/USB_shadow_map”

```
{
```

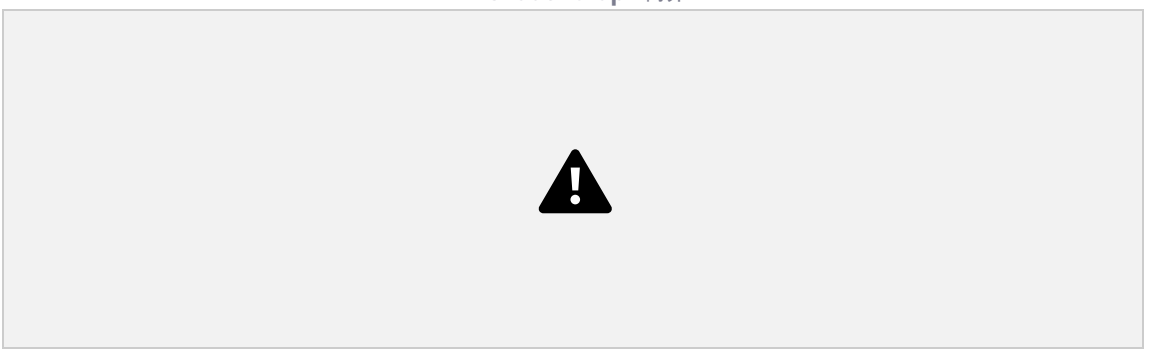
特性{ … }
子着色器

另一件事是，使用此接口创建的着色器很可能无法在其他版本中正确编译。这是因为每次更新中都会添加新功能，有时会影响节点集，如果您使用自定义功能则更是如此。

那么，Shader Graph 是开发着色器的好工具吗？答案是肯定的，对于艺术家来说更是如此。

对于那些使用过 Maya 或 Blender 等 3D 软件的人；Shader Graph 将非常有用，因为它使用的节点系统非常类似于超级着色器和着色器编辑器这允许更直观的着色器创建。

●●●Shader Graph 简介



（图*9.0.1b*）

在介绍本主题之前，请注意 Shader Graph 接口根据其版本的不同而具有功能变化，例如，在撰写本书时，其最新版本是**12.0.0**。

如果您使用此版本创建节点，您可以看到顶点着色器阶段和片段着色器阶段单独出现并独立工作。但是，如果您转到版本**8.3.1,**两个阶段都合并在一个名为掌握，指的是最终着色器输出颜色。

正如您之前提到的，在此接口中创建的着色器很可能无法在其所有版本中编译，事实上，如果您在版本 8.3.1 中创建着色器并更新到版本 12.0.0，则可能会出现功能更改阻止其编译。

调用 .hlsl 文件定制灯并开始添加您的函数，如下所示：

自定义功能●●●

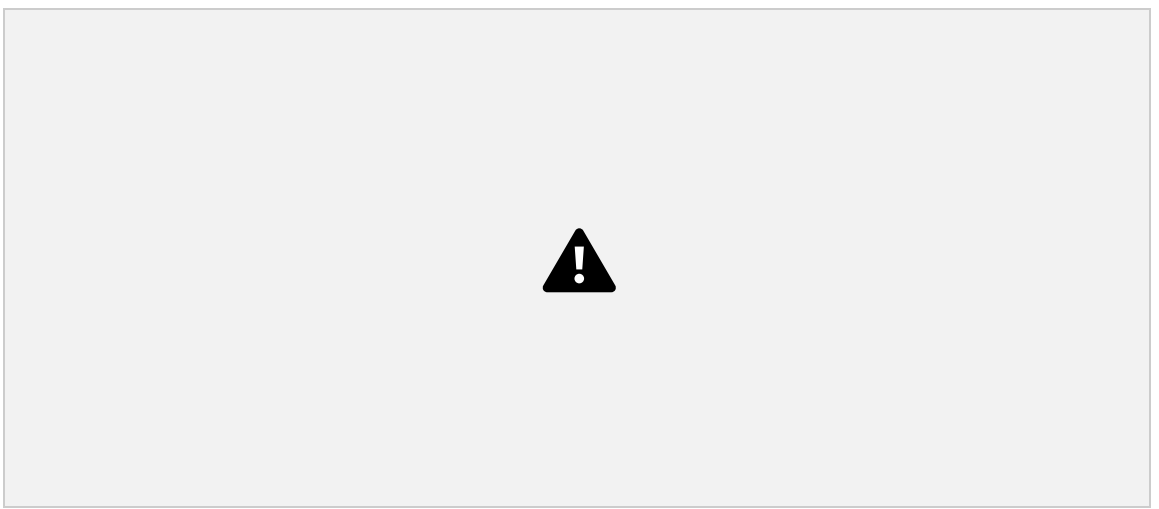
```
// 自定义Light.hlsl
空白自定义Light_float（出半3方向）
{
```

```
        #ifdefSHADERGRAPH_PREVIEW
            方向= 一半3（0,1,0）；
        #别的
            #如果已定义（包括通用照明）
                光主光源=获取主光();
                方向=主光.方向；
            #万一
        #万一
    }
```

在前面的代码块中，您可以推断，如果启用了 Shader Graph 中的预览 (SHADERGRAPH_PREVIEW)，您将在和斧头。否则，如果通用**RP**已定义，则输出方向将与
主光，即场景中的定向光。

与之前的案例不同的是，定制灯函数没有输入；因此，必须在节点配置中添加一个三维向量作为输出。另一方面，值得注意的是，这样的输出是半**3**类型。因此，节点的精度将等于 16 位值，因为默认情况下，它配置为你继承。

接下来，您必须确保拖动或选择**.hlsl**文件在来源位于图形检查器中的框节点设置窗户。您必须确保在函数中使用相同的名称姓名盒子;即CustomLight，否则可能会产生编译错误。

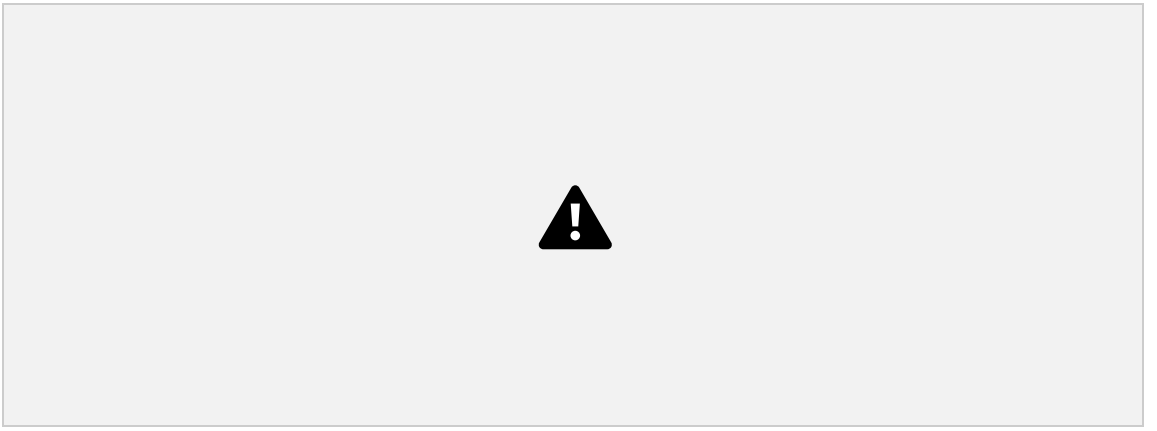


（图 *9.0.7c*。由于方向向量的值，自定义函数节点呈绿色）

由于封闭的变量**_WorldSpaceLightPos**有灯光位置，可以使用节点来复制相同的行为；事实上，操作主光方

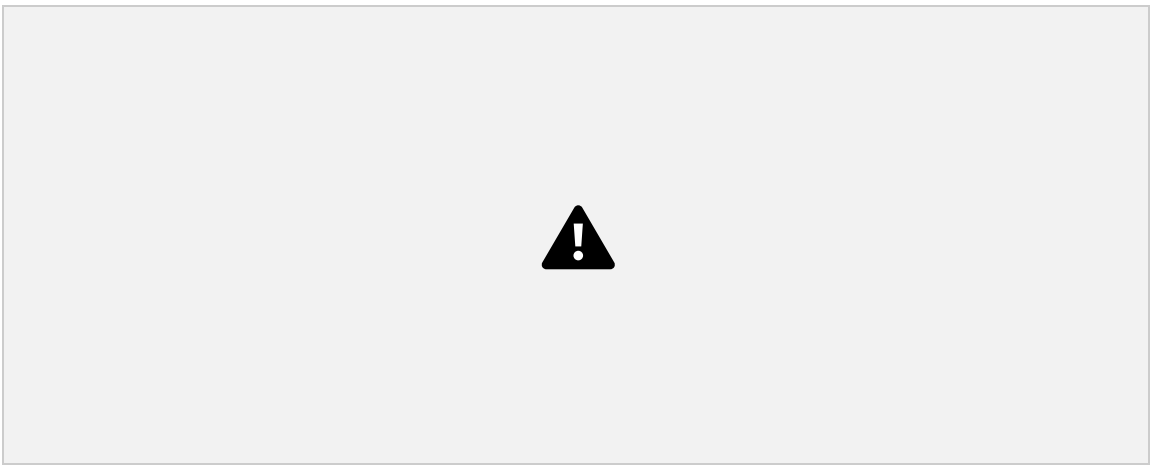
向与变量相同_主光位置包括。您可以通过转到项目的照明**.hlsl**文件。

自定义功能●●●



（图*9.0.7d.D=max(0,N·L)*）

您可以按照第 7.0.3 节中的扩散方案，通过 Shader Graph 中的节点生成相同的行为，如图 9.0.7d 所示。



球体追踪●●●（图*11.0.0b。"Z"*等于*0.0f*）

这等于说，

$$|| X|| - 1 = 0$$

所以，

球体追踪●●●

```
漂浮球体SDF（浮动3p，漂浮半径）
{
    漂浮领域=长度（页）-半径；
```

```
        返回领域；
    }
```

隐式曲面由函数定义，给定空间中的点，表明该点位于曲面内部或外部。

光线从相机穿过像素，直到撞击表面以实现目标。这个概念称为光线投射，这是沿着光线寻找最近物体的过程，因此得名“球体投射”。


```
        // 将值传递给函数
        漂浮t=球体铸造（射线原点，射线方向）；

        //计算平面的空间点
        浮动3p=射线原点+射线方向*t;

        如果（i.hitPos>_边缘）
            丢弃；

        返回山口；
    }
```

看前面的示例，您已将平面相对于相机的距离存储在“t”变量中，然后将每个点存储在变量“p”中。需要将 SDF 平面投影到球体的正面。因此，您必须从 SubShader 禁用剔除。

使用球体追踪实现功能●●●

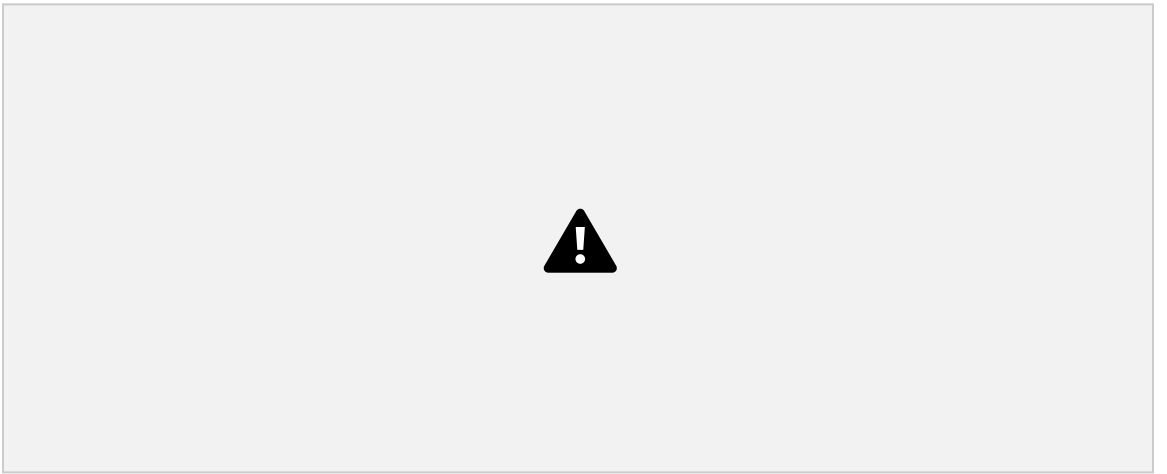
```
    子着色器
    {
        ……
        // 投影球体的两个面
        剔除离开
        ……
        经过
        {
            ……
        }
    }
```

使用**SV_isFrontFace**从语义上讲，您可以将球体的像素投影到其背面，将平面投影到正面。

高清 RP 的特点是其高质量的渲染和与高端平台的兼容性，即个人电脑,游戏机**4**，或者**Xbox One**（向前）。

它还支持**DirectX 11**以及更高版本和着色器模型**5.0**，其中引入了用于图形加速的计算着色器。

打开场景时，某些纹理看起来像图 12.0.1c 中的纹理是很常见的。这是由于生成的配置错误光照贴图创建项目时。



在 **HDRP** 中配置光线追踪●●●（图 *12.0.1c*。墙壁的光照贴图有错误 ）

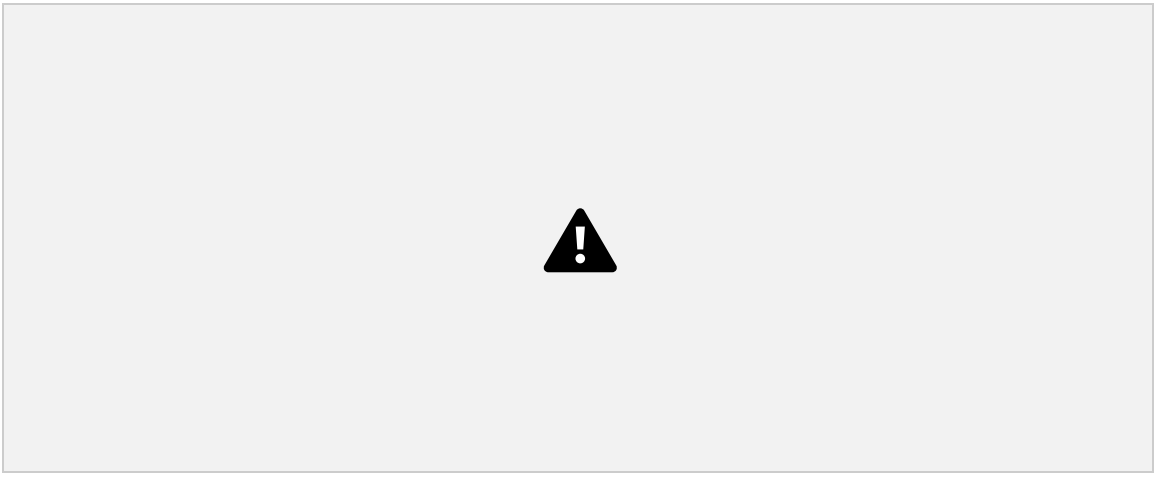
要解决这个问题，必须注意对象的配置。如果您选择任何对象，例如**FR_SectionA_01_LOD0**（墙 ），您会注意到它已被 Unity Inspector 标记为“静态”。

所以必须进入菜单Windows/Rendering/Lighting并执行以下操作：

按“生成照明”按钮的下拉菜单，然后选择“清除烘焙数据”。通过执行此操作，所有光照贴图都将被消除，您将看到默认光照。

接下来，您必须按下“生成照明”按钮。

该过程可能需要几分钟，具体取决于您计算机的容量。然而，纹理和光照最终会正确显示。



在 **HDRP** 中配置光线追踪●●●（图*12.0.1d*。光照贴图已被修正）

请注意，全局照明和其他属性（例如环境光遮挡）正在被烧毁到每个纹理上。因此，如果更改对象的位置，其光照属性将保持其形状并且不会重新计算。

实时执行此过程的唯一方法是激活光线追踪。为此，您必须考虑多种配置，包括 DirectX 12 (DX12)。整个过程可以概括为三个主要步骤：

渲染管线资源。
项目设置。
DirectX 12。

首先进入菜单 Windows / Panels / Project Settings，注意以下类别：

质量。
图形。
HDRP 默认设置。

请注意，Unity 为项目中的每个质量级别创建不同的渲染配置，例如，我们的项目具有三个默认质量级别，可以在质量标签。

高质量。
中等质量。
低质量。

为什么不得到完整的书呢？

Unity 着色器圣经书上有一个**5**星评分

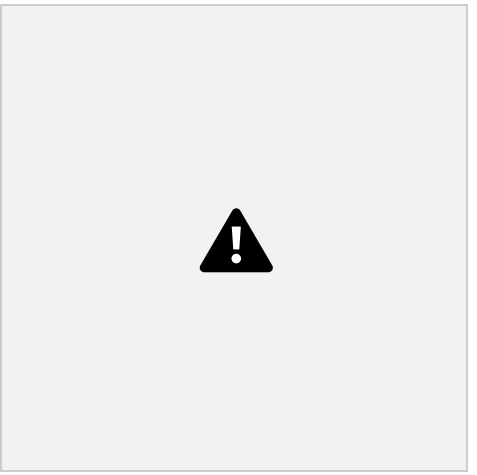
已经结束了**8K**到目前为止的读者！

您愿意加入我们来了解着色器吗？

包含超过 **380** 页适用于游戏开发的信息。

更新永久免费！

在 Gumroad 上购买



去书吧！