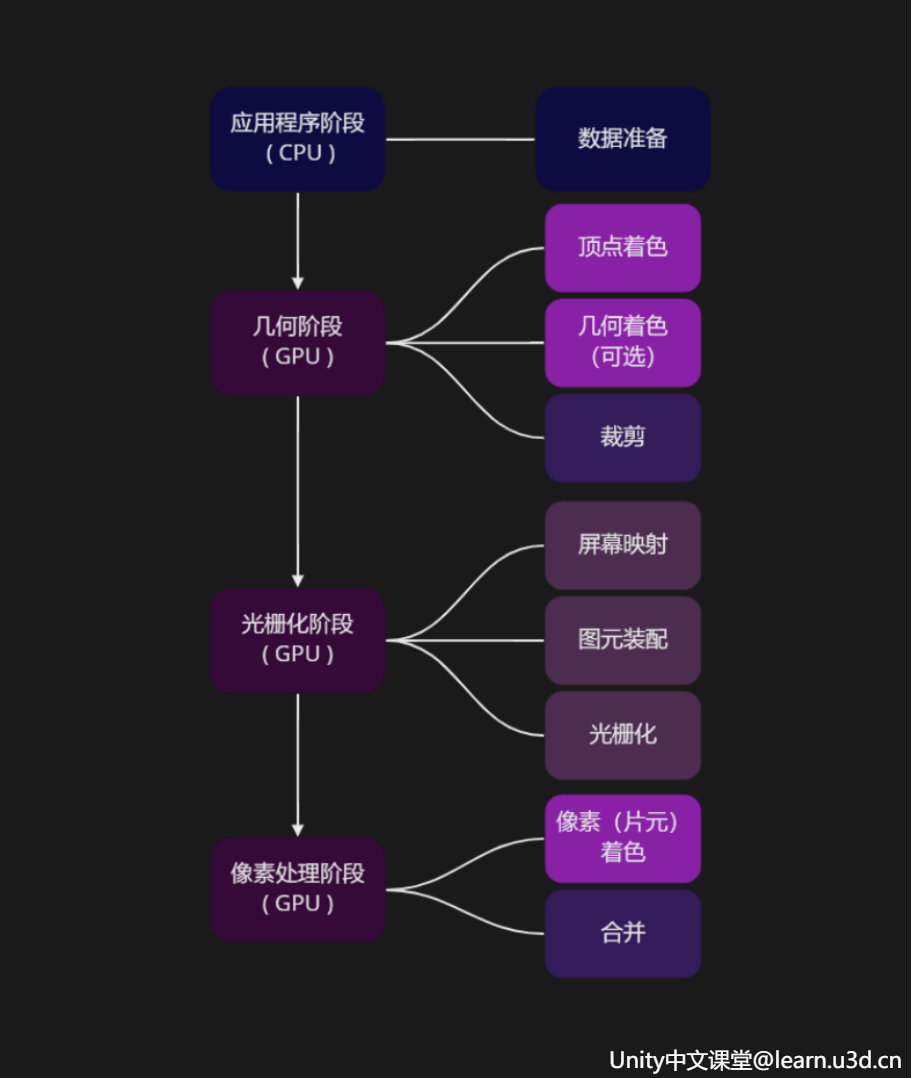
文本版本：

基础概念 - 渲染管线

渲染管线通常来说就是在虚拟相机、三维物体、光源、照明模式、纹理等诸多条件都给定的情况下，生成 或是绘制一幅二维图像的过程。可能不是那么好理解，简单来说，可以把它想象为 美食的制作过程，而这个过程会有很多步骤，这些步骤呢 就是大家经常会听到的渲染阶段。

一般这个过程会分为四个主要阶段：应用程序阶段、几何阶段、光栅化阶段、像素处理阶段。而每个阶段 又会分为很多个部分。



**应用程序阶段 （The Application Stage） CPU**

它最主要是负责数据的准备，也就是准备后面的阶段 所需的数据，像如模型，贴图，光照，相机位置等信息。

-------------------------------------------------------------------------------------------------------------

**几何阶段（The Geometry Stage） GPU**

**顶点着色：**可编程部分，顾名思义，它会对逐个顶点相关的信息进行处理，生成图元，计算并传递给接下来的渲染流程。就像我们平时切菜一样，这个阶段就是为了控制食材的形态，

它对应的则是这里的VertxShader，它的工作主要是计算顶点的位置、法线、纹理坐标，根据材质、纹理、以及光源属性进行顶点光照的计算，平时常见的顶点动画一般就是在这里实现的。

[ **图元：**可以简单理解为它是渲染管线中所有点，线，面的统称 ]

**几何着色：** 可选可编程部分，并非所有GPU都支持 ，它可以把简单的图元拓展成更复杂的形式，通常我们认为，这两大着色器共同构成了 几何阶段的可编程部分。

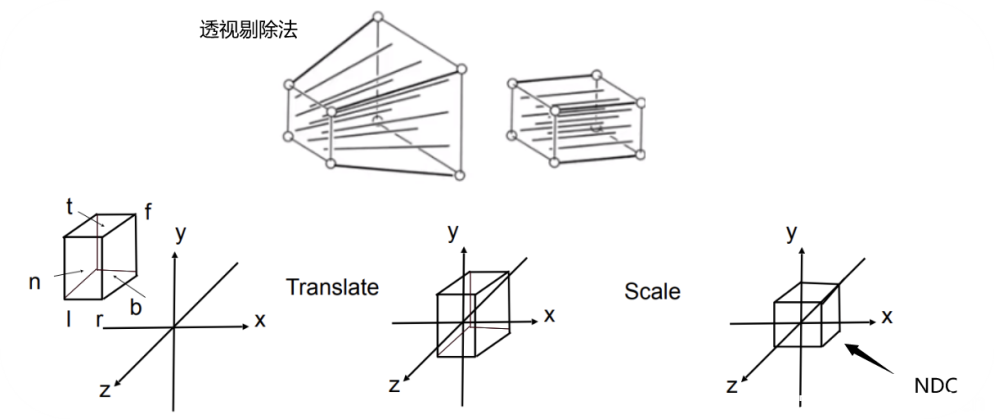
**裁剪 ：** 固定功能硬件实现，对顶点几何两大着色器的输出结果进行处理，它会把完全处于视锥体交界外 以及屏幕窗口外的 几何体部分裁剪掉， 只留下用户能看到的部分，并且对生成的新顶点部分进行插值，输送给接下来的阶段。

-------------------------------------------------------------------------------------------------------------

**光栅化阶段（Rasterization）** **GPU**

**屏幕映射：**经过裁剪之后，硬件会通过透视除法将物体从 裁剪空间 变换 为 标准化设备坐标也叫**NDC**，之后GPU会把得到的NDC空间坐标下的顶点，映射到屏幕空间坐标中，进行图元装配，这一步会计算微分、边方程和其他三角形数据，三角形的朝向剔除就是在这个阶段完成的。

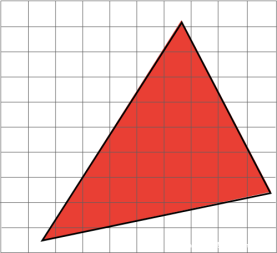
**NDC：**全称Normalized Device Coordinates，一般来说裁剪完成后，会通过透视除法，将物体从裁剪空间 变换为标准化设备坐标NDC，透视除法是将裁剪空间中，顶点的4个分量都除以w分量，从裁剪空间转换到NDC。它是一个长宽高取值范围为[-1,1]的立方体，之所以要转到NDC，是为了方面我们后面进行视口变换把它映射到屏幕空间，不过Unity已经帮我们都完成这些啦，所以具体的数学推算这里不做讲解，有需要的小伙伴这里推荐大家看一下闫令琪老师的 [GAMES101: 现代计算机图形学入门 (ucsb.edu)](https://sites.cs.ucsb.edu/~lingqi/teaching/games101.html)



Vertex Shader -> Clip Space -> 透视除法 -> NDC -> 视口变换 -> Screen Space -> Fragment Shader

**图元装配：**

主要是计算微分（differentials）、边方程（edge equations）和其他三角形数据（顶点属性插值）



**光栅化：**

它会在每个像素点上生成一个片元，如果开启了多重采样抗锯齿，就会对每个像素进行多次采样，产生多个片元，最终进行混合来达到抗锯齿的效果。

[ **片元**：是光栅化之后产生的像素点，因为没有被画到屏幕上，不能被直接称为像素一个像素的最终结果可能是由多个片元来决定的，渲染管线为了细分，就单独创造了片元这个词来描述它，片元只是渲染管线的概念 ]

[ **像素**：则是最后写到图像上的值 ]

-------------------------------------------------------------------------------------------------------------

**像素处理阶段 （ Pixel Processing ）GPU**

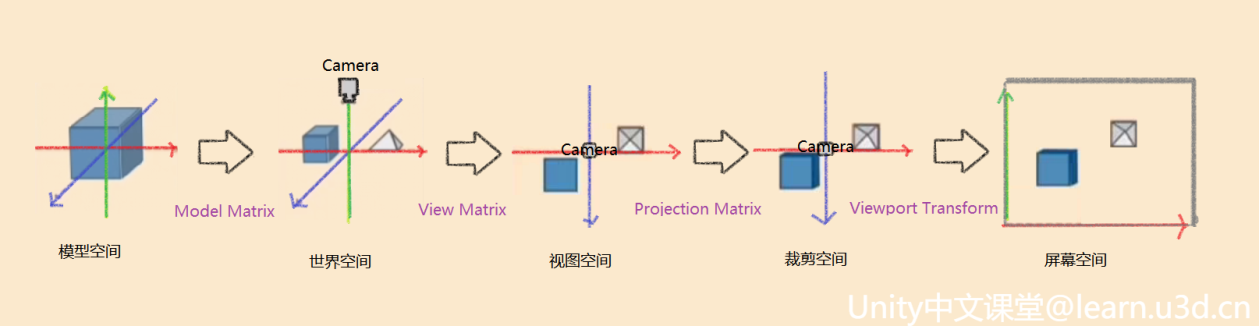
**像素 (片元) 着色器：**可编程部分，它的工作主要是根据顶点的插值属性，进行逐像素计算，因为它需要处理每一个像素，所以这也是最耗时的一个阶段。它的输入输出都是片元数据，输入的数据是 颜色 和纹理坐标，输出的则是计算后所得的每个像素的色彩值，像是逐像素光照、反射、阴影等等更为复杂的效果都是可以在这里实现的，这一步更像是再给我们的食材上色。

**合并：**只可配置不可编程部分，在一系列的测试后会进行合并，所谓的测试则是判断一个像素点最终是否应该被显示在屏幕上，通过测试的颜色会和缓冲区的颜色叠加混合。

-------------------------------------------------------------------------------------------------------------

**坐标空间**

**模型空间** 以物体本身为原点的坐标空间，**世界空间**以世界的(0,0)为原点的坐标空间，**视图空间** 以相机为原点的坐标空间，描述的物体在相机的哪个位置，**裁剪空间** 顶点坐标乘以MVP矩阵之后所在的空间，**屏幕空间** 窗口屏幕上的二维像素坐标空间。



-------------------------------------------------------------------------------------------------------------

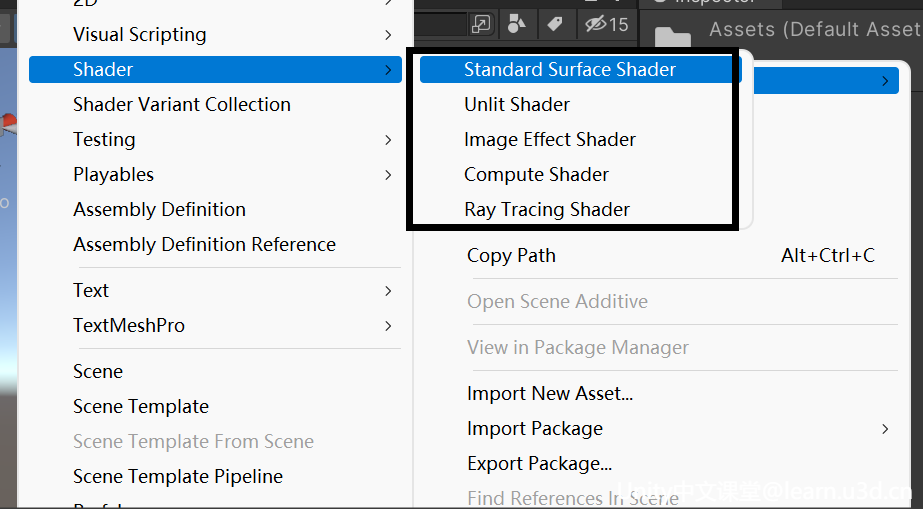
基础概念 - Shader

**Shader**比较学术的百科回答就是用来实现图像渲染的，用来替代固定渲染管线的可编辑程序。其中Vertex Shader（顶点着色器）主要负责顶点的几何关系等的运算，Pixel Shader（像素着色器）主要负责片元颜色等的计算。

那说人话就是可以把渲染流程看作美食的制作流程，一般来说制作美食需要选食材，切菜，炒菜，加调料上色，出锅等流程。我们可以把模型的Mesh看作食材，而Shader在这个流程中主要负责切菜和上色的步骤，切菜控制食材的形态，这就是顶点Shader负责的，加调料上色可以让美食变得更加秀色可餐，这也就是像素Shader负责的。

-------------------------------------------------------------------------------------------------------------

**Unity中的Shader类型**



**Standard Surface Shader**

标准表面着色器，它是一种基于物理的着色系统，可以理解为 它是通过对物理现象的简单模拟，可以实现生活中各种物品的效果，比如石头、木材、玻璃、塑料和金属等等。

**Unlit Shader**

它是最简单的着色器，与 Standard Surface Shader 相比，它去除了冗长的光照公式以及阴影解算，因此得名 Unlit，翻译过来就是无光照，也正因如此，它只由最基础的 Vertex Shader 和 Fragment Shader 组成，最为基础易懂。

**Image Effect Shader**

它其实也是也是顶点片元着色器，不过它主要针对实现各种屏幕后处理效果，那后处理是什么呢？一般像是泛光、调色、景深、模糊等基于最终整个屏幕画面而进行再次处理的就是后处理，这里做个简单的了解即可。

**Compute Shader**

计算着色器，它是在GPU中运行的一段程序，独立于常规渲染管线之外的，它可以直接将GPU作为并行处理器加以利用，从而使GPU不仅具有3D渲染能力，还具有其他的运算能力。一般会在需要大量并行计算的时候使用。

**Ray Tracing Shader**

光线追踪着色器，光线追踪是指从摄像机出发的若干条光线，每条光线会和场景里的物体求交，根据交点位置获取表面的材质、纹理等信息，并结合光源信息计算光照。相对于传统的光栅化渲染，光线追踪可以轻松模拟各种光学效果，如反射、折射、散射、色散等。但由于在进行求交计算时需要知道整个场景的信息，它的计算成本也是非常高的。

-------------------------------------------------------------------------------------------------------------

**ShaderLab**

Unity为我们封装的着色器语言，而目前主流的着色器语言有3种，基于OpenGL的**GLSL** / 基于DX的**HLSL** / NVIDIA公司的**CG**。

GLSL与HLSL分别是基于OpenGL和Direct3D的接口，两者不能混用。而CG则是为了使图形硬件的编程变得和 C语言编程一样方便自由，它本身基于C语言。如果你之前使用过C系语言其中的任意一个，那CG的语法也是比较容易掌握的。但其实由于Microsoft和NVIDIA的相互协作，他们在标准硬件光照语言的语法和语义上达成了一致，所以HLSL和Cg其实可以看为是同一种语言。

而**ShaderLab**则是Unity在HLSL和CG的基础之上封装的只属于Unity的着色器语言，它的灵活性更高，而且不再需要将 Shader 的配置 硬写在引擎代码中，本质是在底层着色语言的基础上，额外提供了声明信息，以数据驱动的方式使我们在渲染管线内自由发挥。