Unity的渲染路径

图形用户界面, 文本

描述已自动生成

前面两个分别是前向渲染路径和延迟渲染路径，后面两个都是已被抛弃的旧版本渲染路径，分别是顶点照明渲染路径和旧版本的延迟渲染路径。

不同的渲染路径也包含不同的标签设置，即Pass中的Tag里写的”LightMode”

LightMode标签支持的渲染路径设置选项有哪些：

图形用户界面, 文本, 应用程序, 电子邮件

描述已自动生成

前向渲染路径

1. 前向渲染路径的原理

每进行一次完整的前向渲染，需要计算渲染该对象的渲染图元，并计算颜色缓冲区和深度缓冲区。利用深度缓冲区决定一个片元是否可见，若可见则更新颜色缓冲区中的颜色值。伪代码表示：

文本

描述已自动生成

Ps：场景中的每个物体对每个逐像素光源都要进行一次完整的渲染流程，如果一个物体在多个逐像素光源的影响区域内，那么该物体将执行多个Pass，每个Pass计算一个逐像素光源的光照结果，然后在帧缓冲中把这些光照结果混合起来得到最终的颜色值，如果有大量逐像素光照会消耗巨大的性能，因此渲染引擎通常会限制每个物体最大的逐像素光照数目。

1. Unity中的前向渲染

Unity中前向渲染路径有三种处理光照的方式：逐顶点处理、逐像素处理、球谐函数。

图形用户界面, 文本, 应用程序, 聊天或短信

描述已自动生成

光源中可以设置渲染模式，设置为Important时，就会被当成逐像素光源处理。

在前向渲染中当渲染一个物体时，Unity会根据场景中各个光源的设置以及光源对这些物体的影响程度对光源进行一个重要程度排序，其中一定数目的光源按逐像素处理，然后最多4个光源按顶点的方式处理，剩下是SH方式处理。

Unity使用的判断规则：

1. 场景中最亮的平行光总是按逐像素处理
2. 渲染模式设置为Not Important的光源会按逐顶点或SH方式处理
3. 渲染模式被设置为Important的光源会按逐像素方式处理
4. 如果根据以上规则得到的逐像素光源小于Quality Setting中的逐像素光源设置数量（Pixel Light Count），会有更多的光源以逐像素方式进行渲染。

前向渲染有两种Pass：Additional Pass和Base Pass

图示

描述已自动生成

对上图的说明：

1．除了设置 Pass 的标签外，还需要分别为 Base Pass 和 Additional Pass 使用 #pragma multi\_compile\_fwdbase 和 #pragma multi\_compile\_fwdadd 编译指令，才可以在相关Pass中得到正确的光照变量，如光照衰减值等。

2．Base Pass 只会执行一次，因此环境光和自发光通常写在 Base Pass里，Base Pass 中渲染平行光默认支持阴影的（如果开启了光源的阴影功能），在 Base Pass 中，可以访问光照纹理(lightmap)，

3．Additional Pass 中渲染的光源没有阴影效果，可以通过使用 #pragma multi\_compile\_ fwdadd\_fullshadows 代替#pragma multi\_compile\_fwdadd ,为点光和聚光 开启阴影，这需要Unity内部使用更多的Shader变种。

4．Additional Pass 的渲染设置，可以开启设置混合模式，因为这样每个 Additional Pass 可与上一次光照结果在帧缓存中进行叠加，从而得到最终的多个光照的渲染效果；若没开启混合模式，那么 Additional Pass 的渲染结果会覆盖掉之前的渲染效果，看起来好像该物体只受该光源的影响。通常选择混合模式为 Blend One One。

5．对于前向渲染，一个Unity Shader通常会定义一个Base Pass (Base Pass也可以定义多次， 例如需要双面渲染等情况）以及一个Additional Pass。 一个Base Pass 只会执行一次（定义了多个Base Pass的情况除外），而每个逐像素光源会执行一次 Additional Pass。

1. 前向渲染内置的光照变量和函数

变量：

图形用户界面, 文本, 应用程序

描述已自动生成

函数：

文本, 应用程序

中度可信度描述已自动生成

顶点照明渲染路径：

它是前向渲染的一个子集，它对硬件配置的要求最少、运算性能最高，但是效果也是最差的，它不支持逐像素才能得到的效果如阴影、法线映射、高精度的高光反射。顶点照明渲染路径只是用到了逐顶点的方式计算光照，如果选择使用顶点照明渲染路径，Unity只会填充逐顶点相关的光照变量，即不可以使用逐像素光照变量。

1.Unity中的顶点照明渲染

顶点照明渲染通常在一个 Pass 中就可以完成对物体的渲染，在这个 Pass 中会计算我们关心的所有光源对该物体的照明，并且这个计算是按逐顶点处理的

2.可访问的内置变量和函数

Ps：Unity中最多在一个顶点照明的Pass中访问到8个逐顶点光源，如果影响该物体的光源数量小于8，那么数组中剩下的光源颜色会设置为黑色

变量：

图形用户界面, 文本, 应用程序

描述已自动生成

函数：

图形用户界面, 文本, 应用程序, 电子邮件

描述已自动生成

延迟渲染路径：

1. 延迟渲染路径的原理

延迟渲染主要包含两个Pass：

第1个Pass：不进行任何光照计算，而是仅仅计算哪些片元是可见的，这主要是通过深度缓冲技术来实现，当发现一个片元是可见的，我们就把它的相关信息存储到G缓冲区中

第2个Pass：利用G缓冲区中的各个片元信息，例如表面法线、视角方向、漫反射系数等进行真正的光照计算

Ps：默认的G缓冲去包含以下渲染纹理：

RT0：格式是ARGB32，RGB通道用于存储漫反射颜色，A通道没有被使用

RT1：格式是ARGB32，RGB通道用于存储高光反射颜色，A通道用于存储高光反射指数部分

RT2：格式是ARGB2101010，RGB通道用于存储法线，A通道没有被使用

RT3：格式是ARGB32(非HDR)或AGRBHalf(HDR)，用于存储自发光+lightmap+反射探针(reflection probes)

伪代码：

文本

描述已自动生成

1. Unity中的延迟渲染

延迟渲染可以支持大量的实时光照，对于延迟渲染路径，它适合在场景中光源数目很多、使用前向渲染会造成性能瓶颈时使用，而且，延迟渲染路径中的每个光源都可以按逐像素的方式处理，但是延迟渲染也存在着以下缺点：不支持真正的抗锯齿(anti-aliasing)；不能处理半透明物体；对显卡有一定要求，如果要使用延迟渲染，显卡必须支持MRT(Multiple Render Targets)、Shader Mode 3.0及以上、深度渲染纹理以及双面的模板缓冲。

可访问的内置变量和函数

变量：

图片包含 Word

描述已自动生成

Unity的光源类型：

1. 平行光

平行光照亮范围没有限制，可以放在场景中任何位置，几何属性只有方向。

1. 点光源

点光源的照亮范围有限制，表示一个由点发出的向所有方向延伸的光（球体），点光源会衰减，随着物体远离点光源，受到的光的强度也会逐渐减小。

1. 聚光灯

照亮空间有限但不是一个球体，而是空间中的一块锥形区域，表示由一个特定位置出发、向特定方向延伸的光，同样随着物体远离聚光灯受到光强逐渐减小，光源是1，边界是0。

前向渲染中处理不同的光源类型

电脑萤幕的截图

描述已自动生成

Base Pass：

文本

描述已自动生成

设置Base Pass的Tag和使用对应的编译指令

文本

描述已自动生成

文本

描述已自动生成

文本

描述已自动生成

片元着色器中首先计算了环境光，在Base Pass中计算一次就可以，自发光也是一样，不过本例中没有自发光。然后计算了平行光，若有多个平行光Unity会选择最亮的平行光传递给Base Pass进行逐像素处理，若没有光源作为全黑处理。由于平行光被认为是没有衰减的，故把代表衰减的变量atten设置为1.0.

Additional Pass：

文本

描述已自动生成

更换Tag，打开混合模式，加入Additional Pass的编译指令

其余代码除了片元着色器部分，都没有修改。Additional Pass处理的光源类型可以是平行光、点光源、聚光灯。

电脑屏幕截图

描述已自动生成

上面代码首先判断了当前处理的逐像素光源类型，使用#ifdef指令是否定义了USING\_DIRECTIONAL\_LIGHT来完成这一步，即判断光源是否是平行光。若为平行光则直接获得光源方向，否则\_WorldSpaceLightPos0.xyz获取点光源或者聚光灯在世界空间下的光源位置，再减去世界空间下的顶点位置就可以得到光源方向。

电脑萤幕画面

描述已自动生成

然后处理光源衰减，仍然通过是否定义USING\_DIRECTIONAL\_LIGHT判断光照类型，若为平行光则没有光照衰减，若不为平行光则继续判断光源类型，按照点光源和聚光灯进行相应的光照衰减处理，如果也没有定义点光源和聚光灯的宏，则按平行光处理。首先通过将顶点位置从世界空间转换为光源空间得到 lightCoord，即顶点相对于点光源的坐标；然后计算 lightCoord 的长度的平方，即 dot(lightCoord, lightCoord).rr；使用这个长度的平方作为纹理坐标（Unity为了避免计算光照衰减带来的很大的开销，选择使用一张纹理作为查找表，以此再片元着色器中得到光源的衰减），从 \_LightTexture0 中采样得到衰减因子；最后将衰减因子乘以 UNITY\_ATTEN\_CHANNEL，得到最终的光照衰减值 atten。

电脑游戏的截图

中度可信度描述已自动生成

Unity的光照衰减

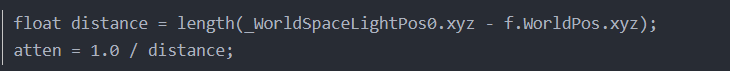
使用纹理衰减的弊端：需要预处理得到采样的纹理，而且纹理的大小也会影响衰减的精度；不直观，把数据存储到查找表中，就无法使用其他数学公式来计算衰减。

1. 用于光照衰减的纹理

Unity内部使用一张\_LightTexture0的纹理计算光源衰减，如果对光源使用了cookie那么衰减查找纹理是\_LightTextureB0。然后就可以通过\_LightMatrix0变换顶点坐标到光源空间，然后进行采样，从采样结果的UNITY\_ATTEN\_CHANNEL分量获取衰减值。

Ps：什么是cookie，"cookie"是指在光源上应用一个纹理，以模拟光源的投影效果。(参考上面例子)

1. 使用数学公式计算衰减



由于Unity没有给出内置衰减值的计算说明，所以对于不同的光源，计算的效果往往不太好

Unity的阴影：

1. 阴影的实现

实时渲染中使用一种叫Shadow Map（阴影映射纹理，本质上是一张深度图，记录了从该位置出发能看到距离它最近的表面位置，也就是深度信息）的技术 ，它会首先把摄像机的位置放在与光源重合的位置上，那么场景中该光源的阴影区域就是那些摄像机看不到的地方。

Unity 使用了一个额外的 Pass 来专门更新光源的阴影映射纹理，这个 Pass 的 LightMode 标签被设置为 ShadowCaster，这个 Pass 的渲染目标不是帧缓存，而是阴影映射纹理(或深度纹理)，Unity 首先把摄像机放置到光源的位置上，然后调用该 Pass，通过对顶点变换后得到的光源空间下的位置，并据此来输出深度信息到阴影映射纹理中。因此，当我们开启了光源的阴影效果后，底层的渲染引擎首先会在当前渲染物体的 Unity Shader 中找到 LightMode 为 ShadowCaster 的 Pass，如果没有，它就会在 Fallback 指定的 Unity Shader 中继续寻找，如果没有找到，该物体就无法向其他物体投影，当找到 LightMode 为 ShadowCaster 的 Pass 后，Unity 会使用该 Pass 来更新光源的阴影映射纹理。

Ps：屏幕空间的阴影映射技术(Screen Shadow Map)，Unity 首先会通过调用 LightMode 为 ShadowCaster 的 Pass 来得到可投射阴影的光源的阴影映射纹理以及摄像机的深度纹理，然后，根据光源的阴影映射纹理和摄像机的深度纹理来得到屏幕空间的阴影图。如果摄像机的深度图中记录的表面深度大于转换到阴影纹理中的深度值，就说明该表面虽然是可见的，但是却处于光源的阴影中。通过这种方式，阴影图就包含了屏幕空间中所有的有阴影的区域。如果我们想要一个物体接收来自其他物体的阴影，只需要在 Shader 中对阴影图进行采样，由于阴影图是屏幕空间下的，因此我们首先需要把表面坐标从模型空间变换到屏幕空间中，然后使用这个坐标对阴影图进行采样即可

Ps：深度图和阴影映射纹理的区别：深度图是一种纹理或图像，记录了场景中每个像素点的深度值（距离摄像机的距离），阴影映射纹理可以看作特殊的深度图，用于存储从光源视角渲染的场景深度信息。

阴影投射的过程：

一个物体接收来自其他物体的阴影，以及向其他物体投射阴影是两个过程：

1.如果想要一个物体接收来自其他物体的阴影，就必须在 Shader 中对阴影映射纹理进行采样，把采样结果和最后的光照结果相乘产生阴影效果

2.如果想要一个物体向其他物体投射阴影，就必须把该物体加入到光源的阴影映射纹理的计算中，从而让其他物体在对阴影映射纹理采样时可以得到该物体的相关信息。Unity 中这个过程是通过执行 LightMode 为 ShadowCaster 的 Pass 来实现的，如果使用了屏幕空间的阴影映射技术，Unity 还会使用这个 Pass 产生一张摄像机的深度纹理

2. 不透明物体的阴影

1.让物体投射阴影

图形用户界面, 文本, 应用程序, 聊天或短信

描述已自动生成

光源中可选择物体投射出的阴影类型，分别为无阴影、硬阴影和软阴影

图形用户界面, 应用程序, Teams

描述已自动生成

在物体的Mesh Renederer组件里的Cast Shadow调整为On，开启后Unity就会把该物体加入到光源的阴影纹理计算中，从而让其他物体在对阴影映射纹理采样时可以得到该物体的相关信息。而Receive Shadow选项则可以选择是否让物体接受来自其他物体的阴影，如果没有开启就不会在内部为该物体计算阴影。

在地板上

中度可信度描述已自动生成

Shader代码中没有LightMode为ShadowCaster的Pass，还能投射阴影的原因是因为



FallBack指定用于回调的内置Specular，内置高光计算的Shader又默认回调VertexLit，最终回调到内置的VertexLit，VertexLit.shader中定义了一个包含ShadowCaster光照类型的Pass。如果把FallBack注释掉就不会产生投影了。

图片包含 徽标

描述已自动生成

可以自己定义LightMode为ShadowCaster的Pass，但没必要，这个Pass的功能通常在多个UnityShader间通用，直接使用FallBack会更方便。

Ps：不仅仅是Specular，Diffuse最终也会调用内置VertexLit。

1. 让物体接受阴影

Base Pass：



添加一个新的内置文件AutoLight.cginc

文本

描述已自动生成

顶点着色器的输出结构体中添加一个内置宏SHADOW\_COORDS，这个宏的作用——声明一个用于对阴影纹理采样的坐标，传入的参数是“下一个可用的插值寄存器的索引值”。

文本

描述已自动生成

顶点着色器中，在返回v2f结构体前使用了一个宏，TRANSFER\_SHADOW，这个宏用于在顶点着色器中计算上一步声明的阴影纹理坐标。

文本

描述已自动生成

使用一个SHADOW\_ATTENUATION宏在片元着色器中计算阴影值。除了刚刚提到的那个宏，SHADOW\_COORDS、TRANSFER\_SHADOW这两个内置宏也可以帮助计算光源的阴影。

这三个宏的作用（需要#include "AutoLight.cginc"）：

SHADOW\_COORDS：声明一个名为 \_ShadowCoord 的阴影纹理坐标变量。

TRANSFER\_SHADOW：调用内置的 ComputeScreenPos 函数来计算 \_ShadowCoord，把顶点坐标从模型空间变换到光源空间后存储到 \_ShadowCoord 中。

SHADOW\_ATTENUATION：使用\_ShadowCoord 对相关纹理进行采样，得到阴影信息。

Ps: 由于这这宏中会使用上下文变量进行相关计算，例如 TRANSFER\_SHADOW 会使用 v.vertex 或 a.pos 来计算坐标，因此为了这些宏能够正常工作，我们需要保证 a2f 结构体中顶点坐标变量名必须是 vertex，顶点着色器的输出结构体必须命名为 v， 且 v2f 中的顶点位置变量必须命名为 pos。

图片包含 地板, 建筑, 瓷砖, 伞

描述已自动生成

1. 统一管理光照衰减和阴影

文本

描述已自动生成

文本

描述已自动生成

图形用户界面, 文本

中度可信度描述已自动生成

文本

描述已自动生成

片元着色器使用内置宏 UNITY\_LIGHT\_ATTENUATION来计算光照衰减，它接受三个参数，将光照衰减和阴影值相乘后的结果存储到第一个参数中（这个传入的变量不需要提前声明），第二个参数是结构体v2f，实际上这个参数被传给了SHADOW\_ATTENUATION用来计算阴影值，第三个参数是顶点世界空间坐标。

Ps：针对不同光源类型、是否启用cookie，Unity声明了多个版本的UNITY\_LIGHT\_ATTENUATION。

这是Additional Pass的片元着色器：

文本

描述已自动生成

由于使用了UNITY\_LIGHT\_ATTENUATION，Base和Additional的代码统一了——不需要在Base Pass中单独处理阴影，也不需要在Additional Pass中判断光源类型来处理光照递减。

Ps：如果希望在 Additional Pass 中添加阴影效果，就需要使用 #pragma multi\_compile\_fullshadows 来代替 #pragma multi\_compile\_fwdadd，这样一来，Unity 也会为额外的逐像素光源计算阴影，并传递给 Shader。

1. 透明度物体的阴影

对于大多数不透明的物体来说，把 Fallback 设置为 VertexLit 就可以得到正确的阴影，但对于透明物体来说，这样会得到错误的结果。

1.透明度测试处理透明度物体阴影

透明度测试的处理比较简单，但如果我们仍然直接使用 VertexLit、Diffuse、Specular 等作为回调，往往无法得到正确的阴影。透明度测试需要在片元着色器中舍弃某些片元，而 VertexLit 中的阴影投射纹理并没有进行这样的操作。

文本

描述已自动生成

文本

描述已自动生成

文本

描述已自动生成

SHADOW\_COORDS宏需要传入的插值寄存器下标是3的原因是，TEXCOORD0、1、2用了前三个插值寄存器，因此现在要传入下标是3的插值寄存器

文本

描述已自动生成

文本

描述已自动生成

如果FallBack是VertexLit的话就会出现错误的阴影效果



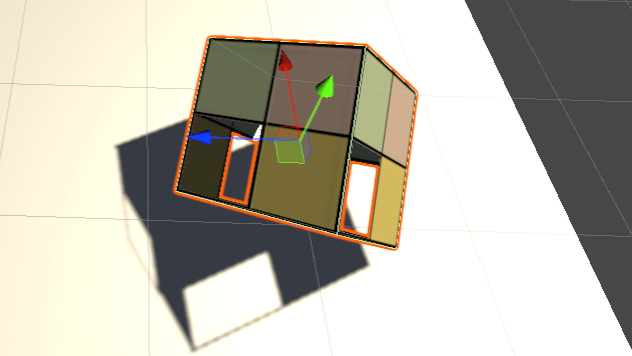
图片包含 看着, 桌子, 蛋糕, 大

描述已自动生成

原因是VertexLit中的含ShadowCaster的Pass没有提供任何的透明度测试的计算，因此会把整个物体的深度信息渲染到深度图和阴影映射纹理中

要想得到正确的阴影，就需要一个进行了透明度测试的ShadowCasterPass





默认情况下把物体渲染到深度图和阴影映射纹理在仅考虑物体的正面，但本例由于一些可以被光照的面背对光源，所以这个阴影也不完全对，需要打开Cube的Two Sided，计算所有面的深度信息

图片包含 看着, 灯光, 蛋糕, 桌子

描述已自动生成

这是正确的

2.透明度混合处理透明度物体阴影

用于半透明物体的阴影，而Unity没有内置任何的计算了透明度混合的ShadowCasterPass

当然，我们可以使用一些 dirty trick 来强制为半透明物体生产阴影，通过把 Fallback 设置为 VertexLit、Diffuse 这些不透明物体使用的 Shader，这样 Unity 就会把它当作不透明的物体来进行阴影的投射。