Contents

0.1 算法部分

本文算法要实现的目标是：用LTC的方式来近似实现多点光源下基于物理的光照效果，同时渲染效率要比常用的BRDF方式更高。

0.1.1 反射率方程

为了实现更真实的渲染效果，如今业界广泛使用如下的反射率方程来实现基于物理的光照渲染：

$$L_o(p,\omega_o) = \int_\Omega f_r(p,\omega_i,\omega_o)L_i(p,w_i)n\cdot\omega_i d\omega_i \tag{1}$$

其中p着色点，n是表面法线，ω_o为观察方向，ω_i为光线入射方向，L_i是来自ω_i方向上的辐射亮度，L_o是着色点在上半球空间接受到的辐射度，f_r是双向反射分布函数（BRDF），描述了反射方向上的辐射照度与入射方向上的辐射照度的比率。当前主流的游戏引擎（如Unity、Unreal等）都在使用基于微面元理论的BRDF模型：

$$f_r(\omega_i,\omega_o) = f_d + \frac{DFG}{4(\omega_o\cdot n)(\omega_i\cdot n)} \tag{2}$$

其中f_d是漫反射系数，D是描述微观面元法线分布的函数，F是描述表面菲涅尔反射的函数，G是描述微观面元之间几何遮挡比率的函数。

所以我们通常使用的反射率方程是：

$$L_o(p,\omega_o) = \int_\Omega (f_d + \frac{DFG}{4(\omega_o\cdot n)(\omega_i\cdot n)})L_i(p,\omega_i)n\cdot\omega_i d\omega_i \tag{3}$$

本文采用的D、F、G函数与Unity引擎内部使用的函数一致。

对于法线分布函数D本文采用GGX分布[?]

$$D = \frac{\alpha^2\chi^+(\theta_h)}{\pi\cos^4\theta_h(\alpha^2+\tan^2\theta_h)^2} \tag{4}$$

其中h表示中间向量，α表示表面粗糙度，θ_h是法向量n与中间向量h的夹角，并且：

$$\chi^+(\theta_h) = \begin{cases} 1, & \theta_h > 0 \\ 0, & \theta_h < 0 \end{cases} \tag{5}$$

对于菲涅尔函数本文采用渲染领域中常用的Schlick近似模型[?]：

$$F = F_0 + (1 - F_0)(1 - (h \cdot \omega_o)^5) \tag{6}$$

其中F_0为入射光垂直表面时的菲涅尔反射率的值，即表面的基础反射率。

对于几何遮挡函数G，本文采用了近年来越来越多的引擎开始使用的更精确的一种遮挡函数[?]：

$$G = \frac{\chi^+(\omega_o \cdot h)\chi^+(\omega_i \cdot h)}{1 + \Lambda(\omega_o) + \Lambda(\omega_i)} \tag{7}$$

其中：

$$\Lambda(m) = \frac{-1 + \sqrt{1 + \alpha^2 \tan^2(\theta_m)}}{2} \tag{8}$$

其中θ_m是向量m和法线n之间的夹角。

对于式-2中的漫反射项f_d，本文采用Disney最新使用的漫反射模型[?]：

$$f_d = \frac{c}{\pi}(1 + (F_D - 1)(1 - cos\theta_l)^5)(1 + (F_D - 1)(1 - cos\theta_v)^5) \tag{9}$$

其中θ_l是ω_i和法线n的夹角，θ_v是ω_o和n的夹角，并且:

$$F_D = 0.5 + 2\alpha \cos^2 \theta_d \tag{10}$$

其中θ_d是入射向量ω_i和中间向量h的夹角。

可以初步看到，渲染时因为要按照上诉公式去分别计算D、F、G项以及f_d，再得到最终的辐射度L_o，所以即使是在一个单点光源的环境下，着色点渲染时所需要的计算量也很大。

0.1.2 使用LTC方法近似BRDF积分

Eric Heitz等人在解决区域光源光照时提出了一种名为线性转换余弦分布（Linear Transform consin）的数学方法[?]，来近似计算BRDF积分。如图1所示： 图中，上面是在某种粗糙度α以及观察向量ω_o下的BRDF分布，分布函数如下：

$$B_s(\omega_i) = \frac{DFG}{4(\omega_o \cdot n)(\omega_i \cdot n)} n \cdot \omega_i \tag{11}$$

下面是一个截断余弦（Clamped Cosine）分布，它的分布函数如下：

$$C(\omega_i = (x, y, z)) = \frac{1}{\pi} max(0, z) \tag{12}$$

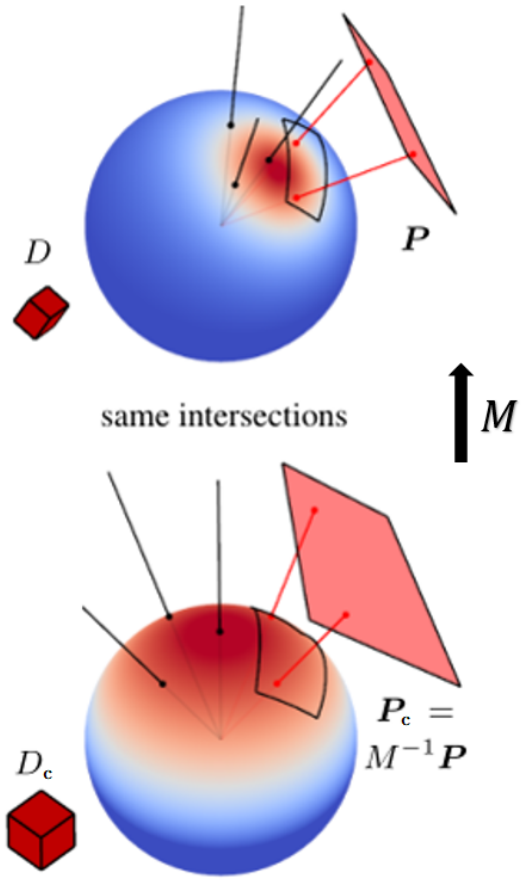

Figure 1: 使用余弦分布乘上矩阵M来近似BRDF分布

其中ω_i是单位向量。

作者提出可以在下面的余弦分布上乘以一个3x3的线性变换矩阵M_s，来近似得到上面的BRDF分布。该矩阵M_s的形式如下：

$$M_s = \begin{bmatrix} a & 0 & b \\ 0 & c & 0 \\ d & 0 & 1 \end{bmatrix} \tag{13}$$

以二维纹理的纹理坐标u作为观察角度θ_o，以纹理坐标v作为粗糙度α，拟合出不同的矩阵M_s。可以将预计算好的M_s的逆矩阵存入二维纹理坐标(u,v)所对应的纹素中。作者提出在BRDF分布上对一个多边形光源区域P的积分，等效于在余弦分布上对另一个多边形区域P_c的积分（多边形P_c是对多边形P乘上

矩阵M_s的逆矩阵之后，得到的新的多边形）：

$$\int_P B_s(\omega)d\omega = \int_{P_c} C(\omega_s)d\omega_s \tag{14}$$

其中ω是在BRDF分布下的光线入射向量，而ω_s是ω乘上矩阵M_s^{-1}作线性变换之后的新的入射向量。

0.1.3 使用LTC方法来计算单点光源光照

将上式-14中的ω看作单位球上的无限小的面积，那么上式中对ω的积分实际上就是在对一个点光源进行光照计算。那么一个着色点接受到来自ω方向上的点光源的镜面辐射度为：

$$L_s = B_s(\omega) = C(\omega_s) \tag{15}$$

其中ω_s是对ω乘上线性余弦变换矩阵M_s的逆矩阵之后得到的新的向量，即：

$$\omega_s = M_s^{-1}\omega \tag{16}$$

但是实际上在式14中无限小的$d\omega$和$d\omega_s$的面积是不一样的（因为$d\omega_s$是$d\omega$作线性变换得到的，线性变换的过程中会改变面积大小）。所以式15是不成立的，还需要乘上两个ω的面积比例：

$$L_s = B_s(\omega) = C(\omega_s)\frac{\partial\omega_s}{\partial\omega} \tag{17}$$

作者在文中已经给出了$\frac{\partial\omega_s}{\partial\omega}$的推导结果，即：

$$B_s(\omega) = C(\omega_s)\frac{\partial\omega_s}{\partial\omega} = C(\omega_s)\frac{|M_s^{-1}|}{||M_s^{-1}\omega||^3} \tag{18}$$

同样的，本文对式9中的Disney漫反射也使用LTC方法拟合出另一个线性余弦变换矩阵M_d。那么一个着色点接受到来自ω方向上的点光源的漫反射辐射度为：

$$L_d = B_d(\omega) = C(\omega_d)\frac{|M_d^{-1}|}{||M_d^{-1}\omega||^3} \tag{19}$$

其中：

$$B_d(\omega) = f_d n \cdot \omega_i \tag{20}$$

$$\omega_d = M_d^{-1}\omega \tag{21}$$

所以，综合式2、式11、式18、式19以及式20，着色点接受到的来自ω方向上的点光源的辐射度为：

$$L_o = f_r L_i n \cdot \omega = (B_s(\omega) + B_d(\omega))L_i = (C(\omega_s)\frac{|M_s^{-1}|}{||M_s^{-1}\omega||^3} + C(\omega_d)\frac{|M_d^{-1}|}{||M_d^{-1}\omega||^3})L_i \tag{22}$$

由式12、式16、式21可以得到：

$$C(\omega_s) = \frac{(M_s^{-1}\omega).z}{||M_s^{-1}\omega||} \tag{23}$$

$$C(\omega_d) = \frac{(M_d^{-1}\omega).z}{||M_d^{-1}\omega||} \tag{24}$$

代入式22可得：

$$L_o = \frac{\omega_s.z}{(\omega_s \cdot \omega_s)^2}|M_s^{-1}|L_i + \frac{\omega_d.z}{(\omega_d \cdot \omega_d)^2}|M_d^{-1}|L_i \tag{25}$$

可以看到，借助LTC方法来计算点光源的辐照度，只需要计算几次向量乘法、平方以及矩阵的行列式就可以了，计算量相当简单（不需要求矩阵的逆，因为本文在预计算的二维纹理中存储的就是M^{-1}，而不是M）。

0.1.4 使用LTC方法来计算多个点光源光照

由式25可得，着色点在多个点光源光照下的辐射照度为：

$$L_o = \sum_{\omega_i}(\frac{\omega_s.z}{(\omega_s \cdot \omega_s)^2}|M_s^{-1}|L_i + \frac{\omega_d.z}{(\omega_d \cdot \omega_d)^2}|M_d^{-1}|L_i) \tag{26}$$

其中$\omega_s = M_s^{-1}\omega_i$、$\omega_d = M_d^{-1}\omega_i$。因为对同一个着色点来说，对任何点光源$M_s^{-1}$和$M_d^{-1}$是定值，所以：

$$L_o = |M_s^{-1}|\sum_{\omega_i}\frac{\omega_s.z}{(\omega_s \cdot \omega_s)^2}L_i + |M_d^{-1}|\sum_{\omega_i}\frac{\omega_d.z}{(\omega_d \cdot \omega_d)^2}L_i \tag{27}$$

而使用BRDF方式来计算多点光源光照的公式如下：

$$L_o = \sum_{\omega_i}(f_d + \frac{DFG}{4(\omega_o \cdot n)(\omega_i \cdot n)})L_i n \cdot \omega_i \tag{28}$$

其中D、F、G、f_d需要分别按照式4、6、7、9来计算。

从式27、28可以看出，本文用LTC的方式来实现多点光源光照，计算量比使用BRDF方式要少很多，从而达到本文要实现的目标：以更快的速率实现多点光源下基于物理的光照效果。除此之外，使用本文LTC渲染点光源的方法，不会因为BRDF公式（DFG以及f_d项）的复杂性而增加渲染时间，也就是说，即使未来BRDF因为要达到更加物理真实的效果而变得更加复杂，使用本文的方法是不会降低渲染帧率的。本文算法对于点光源在基于物理的渲染上具有很好的可扩展性。甚至可以让复杂的点光源PBR光照在移动平台上也能快速实现出较好的效果。

0.1.5 实验结果

本文的实验环境是：Intel(R)Xeon(R)CPU E3-1230 V2@3.30GHz，16GB内存，NVIDIA GeForce GTX 1060 6GB显卡。

在有100个点光源的场景中，使用BRDF方式以及本文的算法在场景粗糙度$\alpha = 0.27$时所实现的渲染结果如图2所示：

当渲染不同数量的光源时，使用BRDF方式以及本文算法的帧率和提升的渲染时间如表1所示：

	100个光源	500个光源	1000个光源	5000个光源
BRDF帧率(fps)	118	23	12	2
本文算法帧率(fps)	285	56	28	5
本文算法时间提升(ms)	4.97	25.62	47.62	255.49

Table 1: 本文算法以及BRDF的渲染效率

(a) 使用BRDF方式渲染的结果

(b) 使用本文算法渲染的结果

Figure 2: 使用BRDF和本文算法结果的对比