Kulla-Conty BRDF

Mengzhu Wang

2024 年 5 月 19 日

1 Overview

微表面模型的 BRDF（Microfacet BRDF）存在一个根本问题，就是忽略了微平面间的多次弹射，这就导致了材质的能量损失，并且当材质的粗糙度越高时，能量的损失会越严重。通过引入一个微表面 BRDF 的补偿项，来补偿光线的多次弹射，能够使得材质的渲染结果可以近似保持能量守恒，这就是 Kulla-Conty BRDF 近似模型。

微表面 BRDF 的公式如下：

$$f(i,o) = \frac{F(i,h)G(i,o,h)D(h)}{4(n,i)(n,o)}$$

F 项使用 Schlick 近似：

$$F = R_0 + (1 - R_0)(1 - \cos\theta)^5$$

D 项使用 GGX 法线分布：

$$D_{GGX}(h) = \frac{\alpha^2}{\pi((n \cdot h)^2(\alpha^2 - 1) + 1)^2} \qquad \alpha = roughness^2$$

G 项使用与 GGX 法线分布匹配的 Smith 模型：

$$G_{Smith}(i,o,h) = G_{Schlick}(l,h)G_{Schlick}(v,h)$$
$$k = \frac{(roughness + 1)^2}{8}$$
$$G_{Schlick}(v,n) = \frac{n \cdot v}{n \cdot v(1 - k) + k}$$

不过实际代码中 `GeometrySchlickGGX` 使用的是 UE4 SIGGRAPH 2013 版本的 $k = (a * a/2)$，并非作业描述中提到的 $(a + 1)^2/8$。

2 Precompute

一条光线如果被遮挡，就相当于会发生另一次 bounce。工业界通过经验去补偿多次反射丢失的能量，其实是创建一个模拟多次反射表面反射的附加 BRDF lobe 来估计。假设 uniform radiance 为

1，一次反射后出射的能量就是求 BRDF 的余弦加权半球积分：

$$\begin{aligned}
E(\mu_o) &= \int_{\Omega^+} f_r(\mu_o, \mu_i, \phi) \cos\theta \mathrm{d}\omega \\
&\xlongequal{\mathrm{d}\omega = \sin\theta \mathrm{d}\theta\mathrm{d}\phi} \int_0^{2\pi}\int_0^{\pi} f_r(\mu_o, \mu_i, \phi) \cos\theta \sin\theta \mathrm{d}\theta\mathrm{d}\phi \\
&\xlongequal{\cos\theta\mathrm{d}\theta = \mathrm{d}\sin\theta} \int_0^{2\pi}\int_0^{1} f_r(\mu_o, \mu_i, \phi) \sin\theta \mathrm{d}\sin\theta\mathrm{d}\phi \\
&\xlongequal{\mu = \sin\theta} \int_0^{2\pi}\int_0^{1} f_r(\mu_o, \mu_i, \phi) \mu_i \mathrm{d}\mu_i\mathrm{d}\phi
\end{aligned}$$

那么光线损失的能量则是 $1 - E(\mu_o)$。由于 BRDF 可逆，入射出射方向的损失都要考虑，因此 BRDF 应为

$$f_{ms}(\mu_o, \mu_i, \phi) = c(1 - E(\mu_i))(1 - E(\mu_o))$$

代入到出射能量公式中推导未知变量：

$$\begin{aligned}
E_{ms}(\mu_o) &= \int_0^{2\pi}\int_0^{1} f_{ms}(\mu_o, \mu_i, \phi) \mu_i \mathrm{d}\mu_i\mathrm{d}\phi \\
&= 2\pi \int_0^{1} c(1 - E(\mu_i))(1 - E(\mu_o)) \mu_i \mathrm{d}\mu_i \\
&= 2\pi c(1 - E(\mu_o)) \int_0^{1} (1 - E(\mu_i)) \mu_i \mathrm{d}\mu_i \\
&= \pi c(1 - E(\mu_o))(1 - E_{avg}) \\
&= 1 - E(\mu_o)
\end{aligned}$$

那么 $c = \frac{1}{\pi(1 - E_{avg})}$，$E_{avg} = 2\int_0^1 E(\mu)\mu \mathrm{d}\mu$。

综上所述，预计算目标是 $E(\mu)$ 和 E_{avg}。其中 $E(\mu)$ 随粗糙度和角度两个变量变化，而 E_{avg} 仅随粗糙度变化。

2.1 Precompute $E(\mu)$

$E(\mu)$ 实际上是通过 cos 加权在半球上采样入射光方向，使用蒙特卡洛方法求解该积分，原式可由积分变为求和：

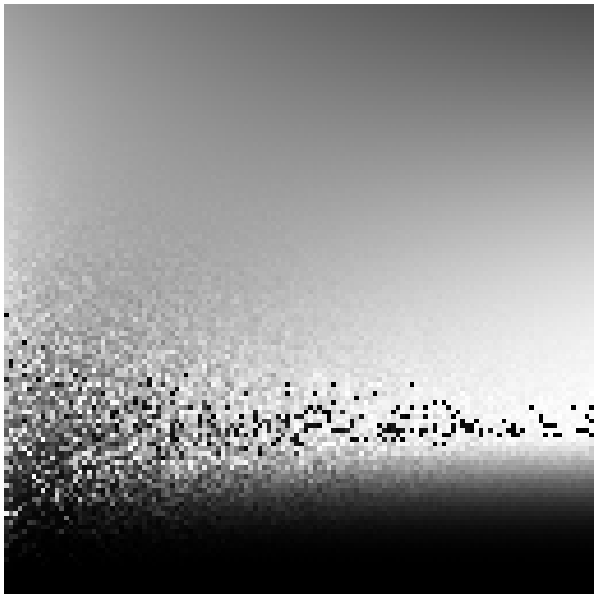
$$E(\mu_o) = \int_{\Omega^+} f_r(\mu_o, \mu_i, \phi) \cos\theta \mathrm{d}\omega \approx \frac{1}{N}\sum_{i=1}^{N} \frac{f_r(\mu_o, \mu_i, \phi)\cos\theta_i}{pdf}$$

```
Vec3f IntegrateBRDF(Vec3f V, float roughness, float NdotV) {
    float A = 0.0;
    float B = 0.0;
    float C = 0.0;
    const int sample_count = 1024;
    Vec3f N = Vec3f(0.0, 0.0, 1.0);
```

```
    samplePoints sampleList = squareToCosineHemisphere(sample_count);
    for (int i = 0; i < sample_count; i++) {
      // TODO: To calculate (fr * ni) / p_o here
      Vec3f L = sampleList.directions[i];
      Vec3f H = normalize(V + L);
      float NdotL = std::max(dot(N, L), 0.0f);
      float F = 1;
      float G = GeometrySmith(roughness, NdotV, NdotL);
      float D = DistributionGGX(N, H, roughness);

      float fr = F * G * D / (4.0 * NdotL * NdotV);
      float pdf = sampleList.PDFs[i];
      A = B = C += fr * NdotL / pdf;
    }

    return {A / sample_count, B / sample_count, C / sample_count};
}
```



横轴表示角度，纵轴表示粗糙度。在粗糙度较低时，计算出的积分值非常小并且有很多噪声。这是因为低粗糙度的微表面材质接近镜面反射材质，即微表面的法线集中分布在几何法线附近，而由采样入射光方向计算出的微表面法向量分布并不会集中在几何法线附近，也就是与实际低粗糙度的微表面法线分布相差很大，因此积分值的方差就会很大。

2.2 Precompute E_{avg}

由于 `IntegrateEmu` 函数传入的参数包括 `Ei` 和 `NdotV`，这两个参数固定，采样点变化也不会导致累加时 `Eavg += Ei * NdotV * 2.0` 的结果变化（为什么是 cos 不是 sin?），因此无需采样直

接返回结果即可。

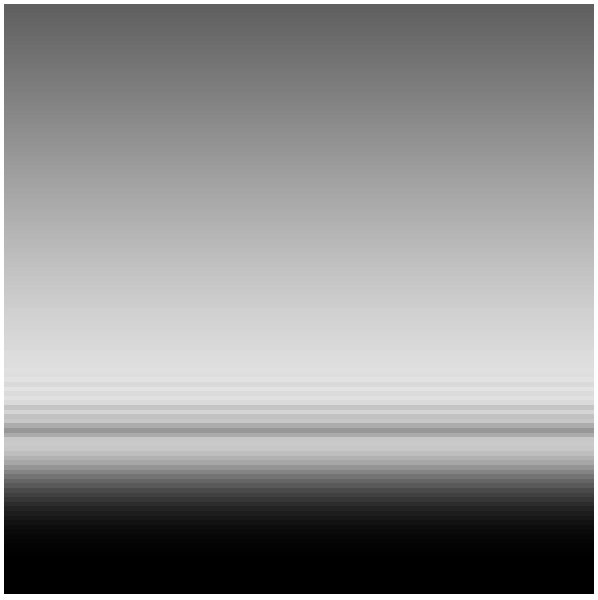
```
Vec3f IntegrateEmu(Vec3f V, float roughness, float NdotV, Vec3f Ei) {
    Vec3f Eavg = Vec3f(0.0f);
    const int sample_count = 1024;
    Vec3f N = Vec3f(0.0, 0.0, 1.0);

    samplePoints sampleList = squareToCosineHemisphere(sample_count);
    for (int i = 0; i < sample_count; i++) {
        Vec3f L = sampleList.directions[i];
        Vec3f H = normalize(V + L);

        float NoL = std::max(L.z, 0.0f);
        float NoH = std::max(H.z, 0.0f);
        float VoH = std::max(dot(V, H), 0.0f);
        float NoV = std::max(dot(N, V), 0.0f);

        // TODO: To calculate Eavg here
        // Eavg += Ei * NdotV * 2.0f;
    }

    return Ei * NdotV * 2.0f;
    // return Eavg / sample_count;
}
```



在粗糙度低时微表面能量损失少，看到的能量多存储结果偏白；在粗糙度高时能量损失多，看到的能量少存储结果偏黑。

3 Real-Time BRDF Compensation

使用与预计算的数据计算模拟经过多次反射后出射的补充 BRDF。该补充 BRDF 与微表面 BRDF 相加即可构成最终我们需要的 Multiple Scattering BRDF。

参考预计算 `lut-gen` 中的微表面模型代码，将 `pbrShader/PBRFragment.glsl` 中微表面 BRDF 的 F、G、D 项近似模型公式补充完成。

根据上一节的推导可得补充 BRDF 项 $f_{ms}(\mu_o, \mu_i)$ 的公式：

$$f_{ms}(\mu_o, \mu_i) = \frac{(1 - E(\mu_o))(1 - E(\mu_i))}{\pi(1 - E_{avg})}$$

同时对于有颜色的材质，会有自然存在的能量损失，这样就需要在之前计算得到的 $f_{ms}(\mu_o, \mu_i)$ 上乘上一个附加的 BRDF 项 f_{add}。有颜色时，第一次有 $F_{avg}E_{avg}$ 的能量被反射，第二次有 $F_{avg}(1-E_{avg})F_{avg}E_{avg}$ 的能量被反射，第 k 次有 $F_{avg}^k(1-E_{avg})^kF_{avg}E_{avg}$ 的能量被反射。这些能量求和得到

$$f_{add} = \frac{F_{avg}E_{avg}}{1 - F_{avg(1-E_{avg})}}$$

最后对于 Kulla-Conty 材质，可以得到的 BRDF 项为：

$$f_r = f_{micro} + f_{add} * f_{ms}$$

```
vec3 MultiScatterBRDF(float NdotL, float NdotV)
{
  vec3 albedo = pow(texture2D(uAlbedoMap, vTextureCoord).rgb, vec3(2.2));

  vec3 E_o = texture2D(uBRDFLut, vec2(NdotL, uRoughness)).xyz;
  vec3 E_i = texture2D(uBRDFLut, vec2(NdotV, uRoughness)).xyz;

  vec3 E_avg = texture2D(uEavgLut, vec2(0, uRoughness)).xyz;
  // copper
  vec3 edgetint = vec3(0.827, 0.792, 0.678);
  vec3 F_avg = AverageFresnel(albedo, edgetint);

  // TODO: To calculate fms and missing energy here
  vec3 f_ms = (1.0 - E_o) * (1.0 - E_i) / PI / (1.0 - E_avg);
  vec3 f_add = F_avg * E_avg / (1.0 - F_avg * (1.0 - E_avg));

  return f_ms * f_add;
}
```

结果图像中上一排是微表面 BRDF+ 补充 BRDF，下一排是微表面 BRDF，从右到左粗糙度逐渐升高。当粗糙度高时，微表面 BRDF 的能量损失会更严重此时渲染结果会偏暗，与上一排有补

充 BRDF 的 Shading Ball 相比没有那么明亮；而在粗糙度低的时候，上下两排对应的 Shading Ball 渲染结果相似。cos 权重采样在低粗糙度的时候误差非常大，使用重要性采样进行预计算的方案在低粗糙度时不存在问题。

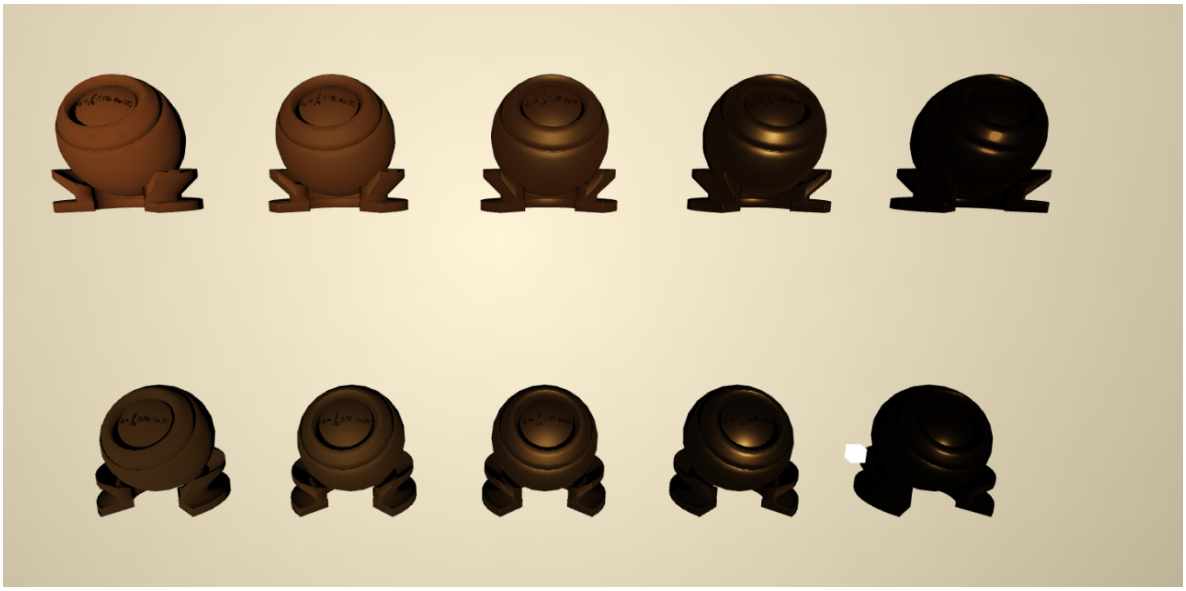


图 1: weighted cos sampling