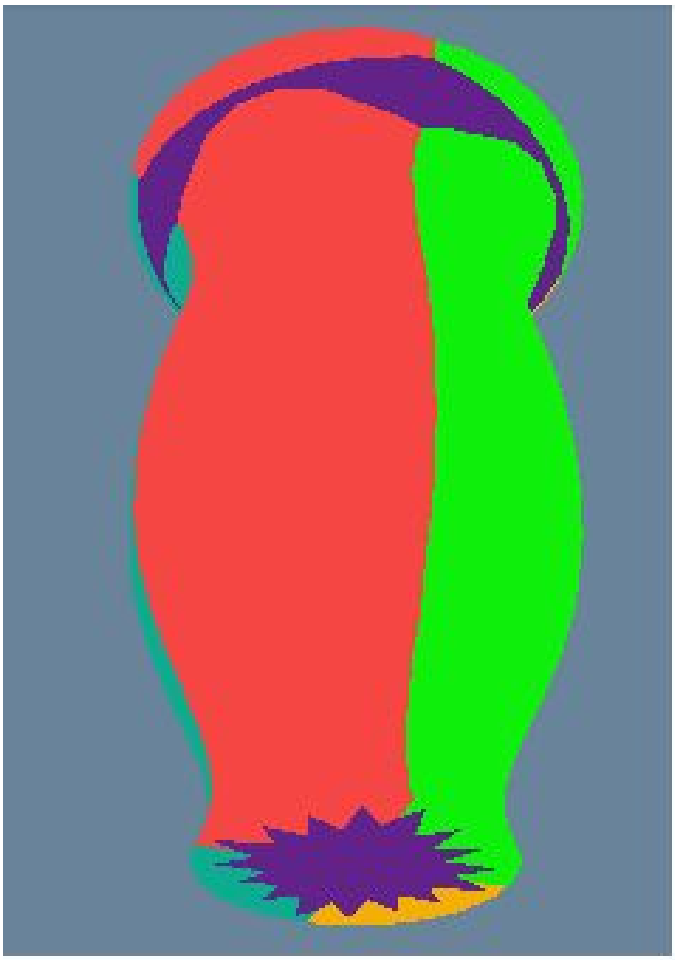
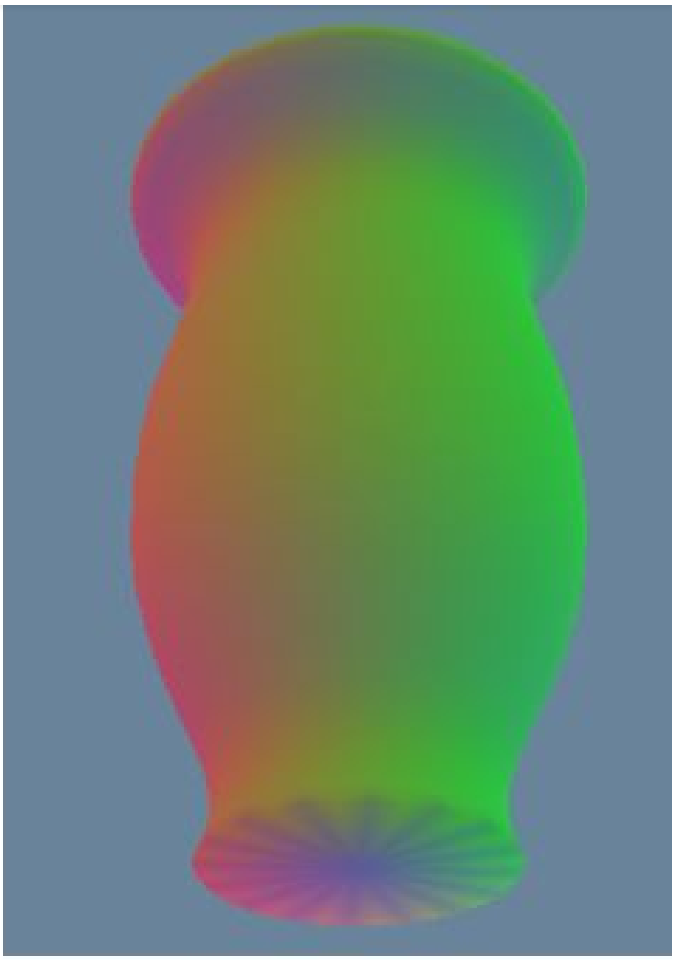


环境映射



球谐光照

球谐光照——辐照度照明


chopper
北京大学 计算机硕士

光照可以简单的分为镜面反射光和漫反射光，基于图像照明技术（Image Based Rendering，IBL）相当于把环境贴图作为光照的来源，可以模拟全局光照中的镜面反射光和漫反射光。

我们讨论过基于图像照明技术在模拟全局镜面光的应用，参见“基于图像的照明”，需要对环境贴图离线预处理。同样，基于照明技术在模拟全局漫反射光的应用中，也需要对环境贴图离线预处理，生成的贴图称为**辐照度环境贴图**（Irradiance Environment Mapping）。用辐照度环境贴图来模拟全局漫反射光存在一定的局限性：离线预处理计算量大，存储开销也大。漫反射光是一

中低频的辐照度信息，这就是基于球谐函数的辐照度照明技术，也就是本篇文章要讨论的。

第1部分“辐照度照明”会介绍一般的辐照度照明技术和该技术的瓶颈，引出基于球谐函数的辐照度照明技术，并举了一个虚幻引擎应用球谐函数光照的例子，也就是间接光缓存技术（Indirect Light Cache）。第2部分“数学推导”阐述了基于球谐函数的辐照度照明技术的理论推导和可行性验证。第3、4部分“投影”和“重建”，阐述了基于球谐函数的辐照度照明技术的两个主要的技术点：投影和重建。任何一个球面函数可以以球谐函数组作为基底进行广义傅里叶展开，生成球谐系数的过程，称为**投影**；根据生成的球谐系数来表示球面函数的过程，称为**重建**。

在github的网站上找到了一个基于球谐函数的辐照度照明的实现，参见justinmeiners/spherical-harmonics。虽然作者的实现存在一定的问题，但是技术实现的结构比较完备，不用从零造轮子，就在它的基础上进行改造验证。

文章目录：

辐照度照明

对于一般的辐照度照明技术，分为两个阶段。首先，需要从环境贴图中提取漫反射光信息，生成一张辐照度环境贴图，如图1所示；接着，在实时渲染的每一个像素点上，根据它的法线信息，从辐照度环境贴图中获取光照信息，进行光照的计算。

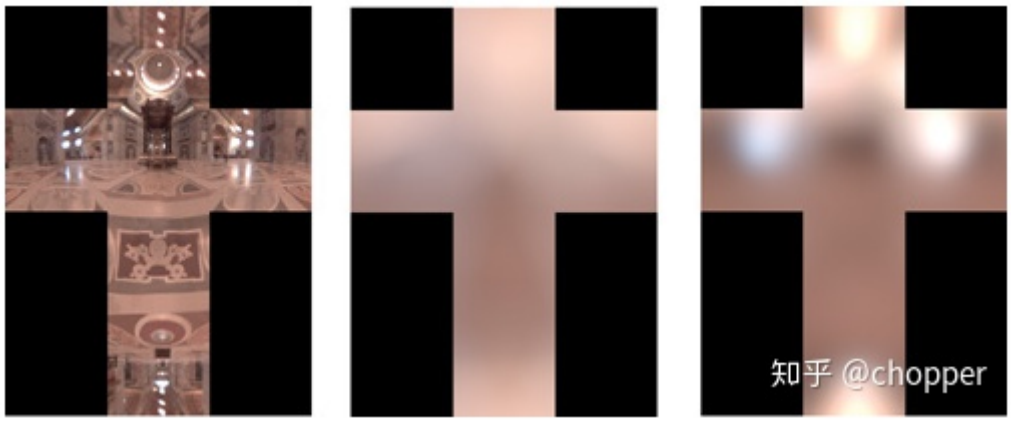


图1. 辐照度环境贴图[3]

生成辐照度环境贴图的算法伪码如下所示，若输出的辐照度贴图的像素个数为M，输入的环境贴图像素个数是N，那么计算的复杂度就是O(MN)。举个例子，输出的辐照度的像素是8 * 8 * 8，

```
diffuseConvolution(outputEnvironmentMap, inputEnvironmentMap){
    for_all (T0: outputEnvironmentMap){
        sum = 0
        N = envMap_direction(T0)
        for_all (T1: inputEnvironmentMap){
            L = envMap_direction(T1)
            I = inputEnvironmentMap[T1]
            sum += max(0, dot(L, N)) * I
            T0 = sum
            return outputEnvironmentMap
        }
    }
}
```

Ravi[2]提出的基于球谐函数的辐照度照明技术，每张辐照度贴图的每个颜色通道只需要存储9个系数，那么，RGB三个通道只需要存储27个系数，能有效地估计辐照度贴图信息。那么，它的离线预处理开销和存储开销也远低于一般的辐照度照明技术。

以虚幻引擎为例，它的间接光缓存技术，本质上就是基于球谐函数的辐照度照明技术。如图2所示，可以在空间内布置一系列的采样点，每个采样点生成一张环境贴图，再从环境贴图中提取投球谐系数，并保存下来。对于静态模型，可以通过烘焙来获取全局漫反射光；对于动态模型，可以通过这种技术来获取全局漫反射光。对处理空间内的动态模型，根据临近几个采样点的球谐系数，插值出新的球谐系数，从而来模拟它所接收到的全局漫反射光。

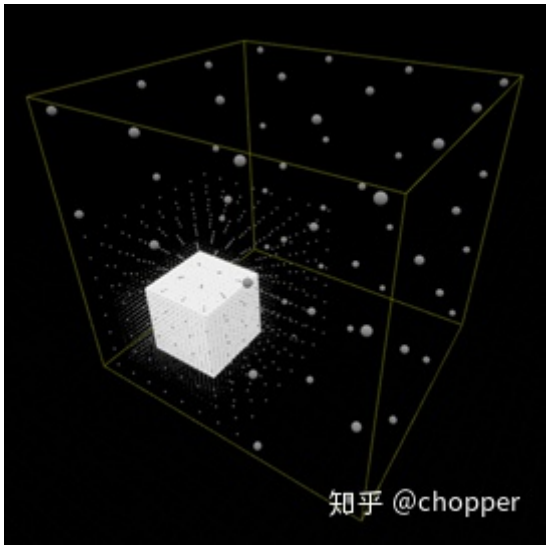


图2. 间接光缓存

很多人可能对辐照度（也称辐射度，英文是Irradiance）和辐射率（英文是Radiance）等名词有点混淆。常说的双向反射分布函数是入射光辐照度与反射光辐射率的比值，也就是说我们需要计

从数学的角度来说，球谐函数具有正交完备性、旋转不变性的性质。若球谐计算的次数 l 越大，即生成的球谐系数越多，就会越好的重建原始的信息，如图3所示。辐照度贴图存储的是光照中的低频信息，低次数的球谐函数就可以较好的重建辐照度贴图中的低频信息，这就是简单的从数学原理上的简单理解。

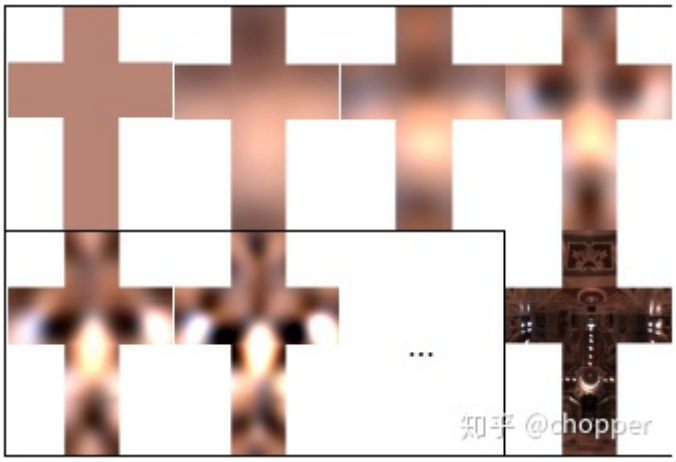
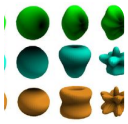


图3. 球谐系数越多，越能重建原始的环境贴图[6]

接下来的“数学推导”部分，将会详细分析该技术方案在数学上的理论推导和可行性验证。

数学推导

chopper：球谐光照——球谐函数
345 赞同 · 21 评论 文章



参见上一篇文章，我们介绍了连带勒让德函数 $P_l^m(x)$ 的递归等式、球谐函数等式以及计算机实现方法，球谐函数表示为

$$Y_l^m(\theta,\varphi)=\begin{cases}\sqrt{2}K_l^m\cos(m\varphi)P_l^m(\cos\theta) & m>0\\ \sqrt{2}K_l^m\sin(-m\varphi)P_l^{-m}(\cos\theta) & m<0\\ K_l^0P_l^m(\cos\theta) & m=0\end{cases}\tag{1}$$

> 其中，K_l^m 是归一化常数，表示为

$$K_l^m=\sqrt{\frac{2l+1}{4\pi}\frac{(l-|m|)!}{(l+|m|)!}}$$

$$f(\theta,\varphi)=\sum_{l=0}^{\infty}\sum_{m=-l}^{l}C_l^m Y_l^m(\theta,\varphi)$$

> 其中，广义傅里叶系数为 C_l^m 。

$$C_l^m=\int_0^{2\pi}\int_0^{\pi}f(\theta,\varphi)\,Y_l^m(\theta,\varphi)\sin\theta d\theta d\varphi$$

生成球谐系数的过程，也称为**投影**。

对于平面上任意一点的入射光辐照度，可以表示为

$$E(\omega_i)=\int_{\Omega}L(\omega)\cos\theta d\omega$$

> 其中， $L(\omega)$ 表示方向为 ω 的辐射率， θ 是入射方向 ω 与平面法线的夹角。

我们的目标就是估计入射光的辐照度 $E(\omega_i)$ ，它是由**光照函数** $L(\omega)$ 和 **传递函数** $max(\cos\theta,0)$ 两部分的乘积组成，我们分别对这两个函数以球谐函数组为基底进行展开。

把光照函数 $L(\omega)$ 展开，表示为

$$L(\omega)=\sum_{l=0}^{\infty}\sum_{m=-l}^{l}L_l^m Y_l^m(\theta,\varphi)$$

光照函数的球谐系数表示为

$$L_l^m=\int_0^{2\pi}\int_0^{\pi}L(\omega)\cdot Y_l^m(\theta,\varphi)\sin\theta d\theta d\varphi \tag{2}$$

设传递函数 $A(\theta)=\cos\theta$ ，这里有一个隐藏的条件： $\theta\in[0,\pi/2]$ 。把传递函数展开，表示为

$l=0\ m=-l$

传递函数的球谐系数可以表示为

$$A_l^m = \int_0^{2\pi} \int_0^{\pi} \cos\theta \cdot Y_l^m(\theta,\varphi) \sin\theta d\theta d\varphi \tag{3}$$

Ravi[1]在它的论文中，推导出 $E(\omega_i)$ 可以表示为

$$E(\omega_i) = \sum_{l=0}^{\infty} \sum_{m=-l}^{l} \sqrt{\frac{4\pi}{2l+1}} L_l^m A_l^0 Y_l^m(\theta,\varphi) \tag{4}$$

那么，设双向反射分布函数为 $brdf(\omega_i,\omega_o)$，那么，最终的光照计算公式就可以表示为

$$B(\omega_o) = brdf(\omega_i,\omega_o) \cdot E(\omega_i) \tag{5}$$

等式（4）（5）有重要的意义，**表示只需要计算出光照函数和传递函数的球谐系数，通过两种系数的内积，就可以计算出环境贴图对模型的漫反射光贡献。**

光照函数的球谐系数计算可以放到下一部分去讨论，先考虑传递函数的球谐系数推导。

由于

$$\int_0^{2\pi} \cos\varphi d\varphi = 0, \int_0^{2\pi} \sin\varphi d\varphi = 0,$$

易知，对于任意的 $m \neq 0$ 的情况下，有 $A_l^m = 0$，所以我们只需要计算 A_l^0 即可，后面统一用 A_l 来表示 A_l^0。令 $x = \cos\theta$，代入等式（3），则有

$$A_l = 2\pi \int_0^1 x \cdot Y_l^0(x) dx$$

又由于 $P_1^0(x) = x$ 和 $Y_l^0(x) = K_l^0 P_l^0(x)$，则有

$$A_l = 2\pi \sqrt{\frac{2l+1}{4\pi}} \int_0^1 P_1^0(x) \cdot P_l^0(x) dx$$

正交性，表示为

$$\int_{-1}^{1} P_k^m(x) \cdot P_l^m(x)\, dx = \frac{2(l+m)!}{(2l+1)(l-m)!}\delta_{k,l}$$

> 其中，当$k=l$时，$\delta_{k,l}=1$，否则，$\delta_{k,l}=0$。

奇偶性，表示为

$$P_l^m(-x) = (-1)^{l+m} P_l^m(x)$$

> 当$m=0$的情况下，若l为奇数，则$P_l^0(x)$就是奇函数；若l为偶数，则$P_l^0(x)$就是偶函数。

回过正题，当$l \succ 1$时，由连带勒让德函数的奇偶性质，可知，当l为奇数时，$P_l^0(x)$为奇函数，当l为偶数时，$P_l^0(x)$为偶函数。

当$l \succ 1$且l为奇数时，由连带勒让德函数的正交性，可知

$$\int_0^1 P_1^0(x) \cdot P_l^0(x)\, dx = \frac{1}{2}\int_0^1 P_1^0(x) \cdot P_l^0(x)\, dx = 0$$

即当$l \succ 1$且l为奇数时，有

$$A_l = 0 \tag{6}$$

当$l \succ 1$且l为偶数时，有

$$A_l = 2\pi\sqrt{\frac{2l+1}{4\pi}}\frac{(-1)^{\frac{n}{2}-1}}{(n+2)(n-1)}\frac{n!}{2^n\left(\frac{n}{2}!\right)^2} \tag{7}$$

此外，很容易推导出A_0和A_1的值，为

$$A_0 = \frac{1}{2}\sqrt{\pi}, A_1 = \sqrt{\frac{\pi}{3}} \tag{8}$$

$$A_0 = \frac{1}{2}\sqrt{\pi} \approx 0.8862$$
$$A_1 = \sqrt{\frac{\pi}{3}} \approx 1.0233$$
$$A_2 = \frac{\sqrt{5\pi}}{8} \approx 0.4954$$
$$A_3 = 0$$
$$A_4 = -\frac{\sqrt{9\pi}}{48} \approx 0.1108$$

Ravi[1]在它论文中，绘制了一张次数 l 变化时，系数 A_l 随之变化的曲线图，如图4所示，可以发现，随着次数 l 变大，系数 A_l 会快速衰减。

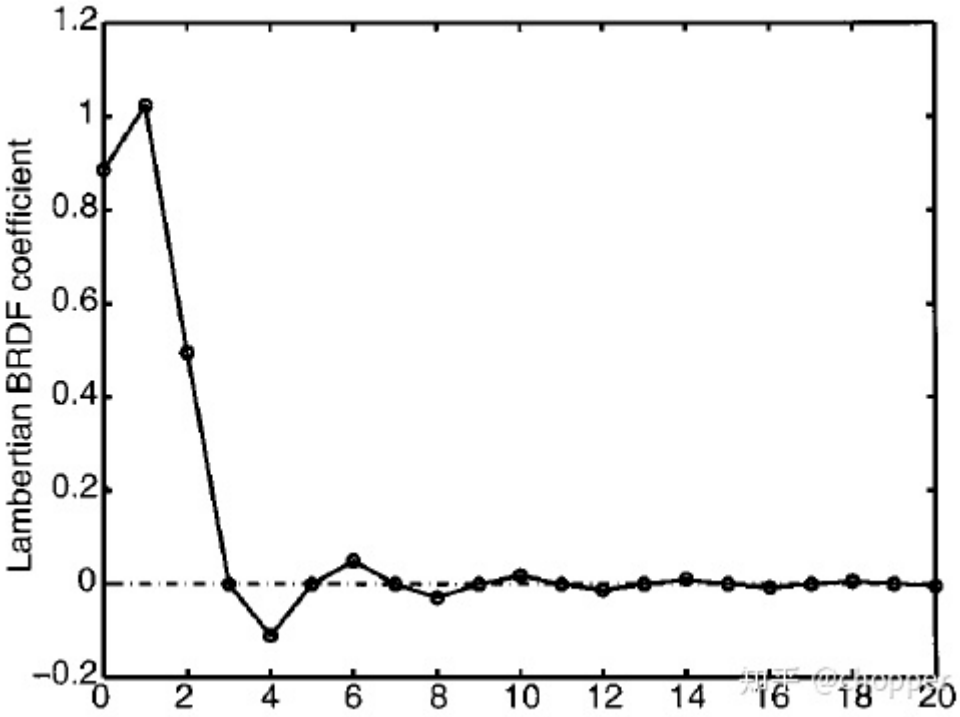


图4. 随着次数l变大，传递函数的球谐系数快速衰减。

再结合等式（4），可以得到结论：随着 l 的变大，由于 A_l 衰减的较快，所以对结果的贡献会越小，所以当我们只需要重建低频信息时，就不需要很高的次数 l，这就是Ravi[2]核心思想的来源。Ravi[2]提出，**只需要前3次，共9个球谐系数，就可以较好的重建环境贴图中的低频辐照信息。**

至此，本部分得到三条重要的结论：

1. 根据等式（4）（5），环境贴图的漫反射光的计算可以分解为光照函数和传递函数的球谐系数的内积和；
2. 光照函数的球谐系数可以根据等式（2）来计算；

为了方便后面的技术阐述，这里给出球谐函数的前三次的函数，表示为

$$Y_0^0 = \frac{1}{2}\sqrt{\frac{1}{\pi}}$$
$$Y_1^{-1} = -\frac{1}{2}\sqrt{\frac{3}{\pi}}y$$
$$Y_1^0 = \frac{1}{2}\sqrt{\frac{3}{\pi}}z$$
$$Y_1^1 = -\frac{1}{2}\sqrt{\frac{3}{\pi}}x$$
$$Y_2^{-2} = \frac{1}{2}\sqrt{\frac{15}{\pi}}yx$$
$$Y_2^{-1} = -\frac{1}{2}\sqrt{\frac{15}{\pi}}yz$$
$$Y_2^0 = \frac{1}{4}\sqrt{\frac{5}{\pi}}\left(3z^2 - 1\right)$$
$$Y_2^1 = -\frac{1}{2}\sqrt{\frac{15}{\pi}}zx$$
$$Y_2^2 = \frac{1}{4}\sqrt{\frac{15}{\pi}}\left(x^2 - y^2\right)$$

投影

这部分要解决的问题是，给定环境贴图，将它投影至球谐基底，如何计算出投影的球谐系数，也就是光照函数的球谐系数。

计算光照函数的球谐系数，它的积分形式，如等式（2）所示，我们可以把它转为求和的形式来表示，就是

$$L_l^m = \sum_{k=0}^{n} L_k\left(\omega\right) \cdot Y_l^m\left(\theta, \varphi\right) d\omega \tag{9}$$

> 其中，$L_k\left(\omega\right)$ 表示环境贴图中的纹素信息，n 表示环境贴图的纹素个数，$d\omega$ 表示对应纹素的立体角。

在光照函数的球谐系数计算中，如等式（9）所示，有两个技术问题：

- 如何计算纹素的立体角；
- 如何根据环境贴图的纹理坐标计算法线信息。

我们采用的环境贴图是**立方体贴图**（Cubemap）。

Rory[5]给出了任意一个纹素的立体角计算方法，如下所示，需要采用双精度double来表示，float格式误差会比较大，容易导致所有纹素的立体角之和不等于4π。

```
static double AreaElement(double x, double y)
{
    return atan2(x * y, sqrt(x * x + y * y + 1));
}

double TexelCoordSolidAngle(double a_U, double a_V, unsigned int a_Size)
{
    //scale up to [-1, 1] range (inclusive), offset by 0.5 to point to texel center.
    double U = (2.0f * ((double)a_U + 0.5f) / (double)a_Size) - 1.0f;
    double V = (2.0f * ((double)a_V + 0.5f) / (double)a_Size) - 1.0f;

    double InvResolution = 1.0f / a_Size;

    // U and V are the -1..1 texture coordinate on the current face.
    // Get projected area for this texel
    double x0 = U - InvResolution;
    double y0 = V - InvResolution;
    double x1 = U + InvResolution;
    double y1 = V + InvResolution;
    double SolidAngle = AreaElement(x0, y0) - AreaElement(x0, y1) - AreaElement(x1, y0

    return SolidAngle;
}
```

简单描述下立体角的计算推导过程，更详细的介绍可以参见Rory[5]。如图5所示，把一个立体贴图的一个面置于(0, 0, 1)的位置，那么它表示的范围就缩放到了[-1, 1]之间，接下来我们就要计算它在球面上投影的面积，就是我们要算的立体角的大小。

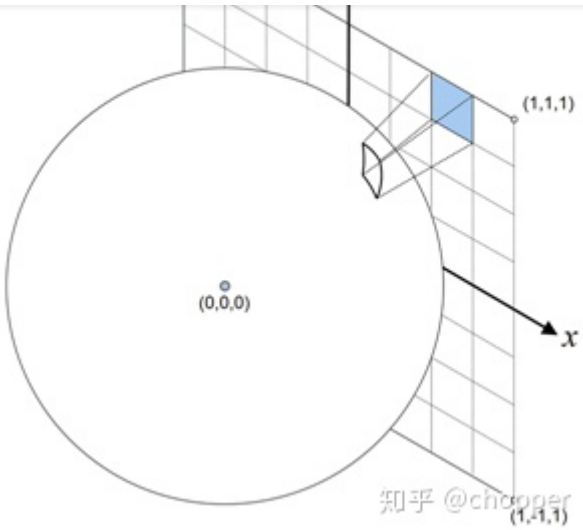


图5. 纹素在球面上投影的立体角[5]

球中心是(0, 0, 0)，对于贴图上任意一个纹素(u, v)，它的坐标可以表示为 $(x, y, 1)$，其中

$$x = 2.0\frac{u + 0.5}{size} - 1.0, y = 2.0\frac{v + 0.5}{size} - 1.0$$

那么，由球中心指向贴图纹理的向量，与球的交点表示为

$$\vec{p} = \frac{(x, y, 1)}{\sqrt{x^2 + y^2 + 1}}$$

计算 $\vec{p}$ 在x和y方向上的偏导，可得

$$\frac{\partial \vec{p}}{\partial x} = \frac{(y^2 + 1, -xy, -x)}{(x^2 + y^2 + 1)^{\frac{3}{2}}}, \frac{\partial \vec{p}}{\partial y} = \frac{(-xy, x^2 + 1, -y)}{(x^2 + y^2 + 1)^{\frac{3}{2}}}$$

偏导的几何意义是球面上的两条切线，对应在球面上的面积就可以表示为

$$dA = \left\| \frac{\partial \vec{p}}{\partial x} \times \frac{\partial \vec{p}}{\partial y} \right\| = \frac{1}{(x^2 + y^2 + 1)^{\frac{3}{2}}}$$

那么，从[0, 0]到[s, t]的纹理，在球面上的投影面积就表示为

$$(x^2+y^2+1)^2$$

对于任意一个纹素，如图6所示，它的立体角就可以表示为：

$$S = f(A) - f(B) + f(C) - f(D)$$

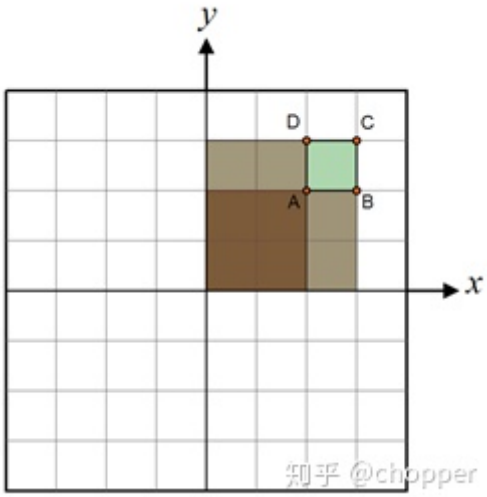


图6. 立体角的计算[5]

接着，要解决立方体贴图的纹理坐标与法线信息的对应关系，以OpenGL的规范为例，参见ARB_texture_cube_map，容易得到一个贴图纹理与法线的对应关系，实现源码可以表示为

```
static inline void uv_to_cube(double u, double v, int face, double* out_dir)
{
    switch (face)
    {
        case CUBE_FACE_RIGHT:
            out_dir[0] = 1.0f;
            out_dir[1] = -v;
            out_dir[2] = -u;
            break;
        case CUBE_FACE_LEFT:
            out_dir[0] = -1.0f;
            out_dir[1] = -v;
            out_dir[2] = u;
            break;
        case CUBE_FACE_TOP:
            out_dir[0] = u;
            out_dir[1] = 1.0f;
            out_dir[2] = v;
```

```
            out_dir[0] = u;
            out_dir[1] = -1.0f;
            out_dir[2] = -v;
            break;
        case CUBE_FACE_BACK:
            out_dir[0] = u;
            out_dir[1] = -v;
            out_dir[2] = 1.0f;
            break;
        case CUBE_FACE_FRONT:
            out_dir[0] = -u;
            out_dir[1] = -v;
            out_dir[2] = -1.0f;
            break;
    }
}
```

至此，投影相关的主要技术问题都已经解决了，结合等式（9），就可以实现光照球谐系数的计算，表示为

```
#define SH_COUNT 9

static inline double sh_eval_9(int i, double x, double y, double z)
{
    switch (i)
    {
        case 0:
            return 0.5 * sqrt(1.0 / M_PI);
        case 1:
            return -y * 0.5 * sqrt(3.0 / M_PI);
        case 2:
            return z * 0.5 * sqrt(3.0 / M_PI);
        case 3:
            return -x * 0.5 * sqrt(3.0 / M_PI);
        case 4:
            return x * y * 0.5 * sqrt(15.0 / M_PI);
        case 5:
            return -y * z * 0.5 * sqrt(15.0 / M_PI);
        case 6:
            return (3.0 * z*z - 1.0) * 0.25 * sqrt(5.0 / M_PI);
        case 7:
            return -x * z * 0.5 * sqrt(15.0 / M_PI);
```

```
        default:
            assert(0);
            return 0;
    }
}

static inline void normalize(double* dir, int n)
{
    double length_sqr = 0.0f;
    for (int i = 0; i < n; ++i)
        length_sqr += dir[i] * dir[i];

    double scale = 1.0 / sqrt(length_sqr);
    for (int i = 0; i < n; ++i)
        dir[i] *= scale;
}

static double surface_area(double x, double y)
{
    return atan2(x * y, sqrt(x * x + y * y + 1.0));
}

void sh_integrate_cubemap(const unsigned char** face_data,
                          unsigned int width,
                          unsigned int height,
                          unsigned int components_per_pixel,
                          ShChannel* out_channels)
{
    // zero out coeffecients for accumulation
    for (int comp = 0; comp < components_per_pixel; ++comp)
    {
        for (int s = 0; s < SH_COUNT; ++s)
            out_channels[comp].coeffs[s] = 0.0;
    }

    double sum = 0.0;
    assert(width == height);
    for (int face = 0; face < CUBE_FACE_COUNT; ++face)
    {
        for (int y = 0; y < height; ++y)
        {
            for (int x = 0; x < width; ++x)
            {
```

```
                double py = (double)y + 0.5;
                // normalize into [-1, 1] range
                double u = 2.0 * (px / (double)width) - 1.0;
                double v = 2.0 * (py / (double)height) - 1.0;

                // calculate the solid angle
                double d_x = 1.0 / (double)width;
                double x0 = u - d_x;
                double y0 = v - d_x;
                double x1 = u + d_x;
                double y1 = v + d_x;
                double d_a = surface_area(x0, y0) - surface_area(x0, y1) - surface_are
                sum += d_a;
                // find vector on unit sphere
                double dir[3];
                uv_to_cube(u, v, face, dir);
                normalize(dir, 3);

                size_t pixel_start = (x + y * width) * components_per_pixel;

                for (int s = 0; s < SH_COUNT; ++s)
                {
                    double sh_val = sh_eval_9(s, dir[0], dir[1], dir[2]);

                    for (int comp = 0; comp < components_per_pixel; ++comp)
                    {
                        double col = face_data[face][pixel_start + comp] / 255.0;
                        out_channels[comp].coeffs[s] += col * sh_val * d_a;
                    }
                }
            }
        }
    }
    printf("sum: %lf\n", sum);
}
```

举个例子，如图7所示的环境贴图，对它离线处理，生成9个球谐系数如下所示，RGB每个通道各有9个常数。

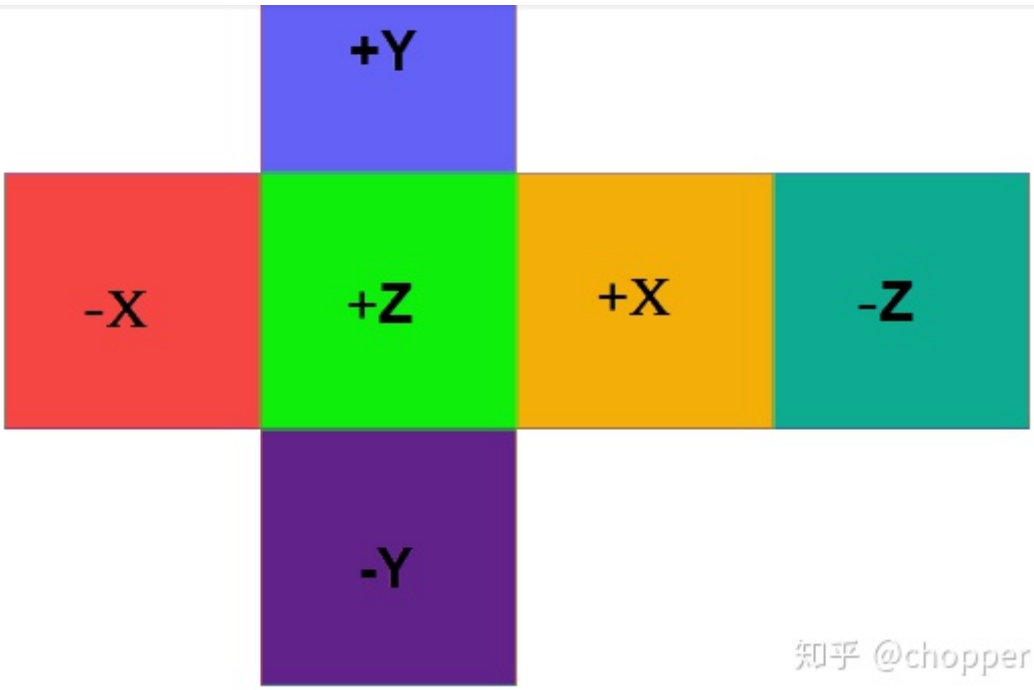


图7. 环境贴图

```
const float sh9_red[] = { 1.649656, -0.010007, 0.003336, 0.006671, -0.000000, 0.000000
const float sh9_green[] = { 1.818793, -0.210143, 0.226821, -0.346903, 0.000000, 0.0000
const float sh9_blue[] = { 1.422597, -0.360245, -0.443635, 0.196801, -0.000000, 0.0000
```

在下一部分的内容阐述中，就可以利用这27个球谐系数来重建环境贴图的辐照度，从而实现辐照度照明。

重建

最后，我们考虑已知环境贴图投影至球谐基底生成的球谐系数，如何重建辐照度信息。

令

$$\hat{A}_l = \sqrt{\frac{4\pi}{2l+1}} A_l$$

则等式（4）转化为

$$E(\omega_i) = \sum_{l=0}^{\infty} \sum_{m=-l}^{l} L_l^m \hat{A}_l Y_l^m(\theta, \varphi) \quad (10)$$

$$\hat{A}_0 = \pi$$
$$\hat{A}_1 = \frac{2}{3}\pi$$
$$\hat{A}_2 = \frac{1}{4}\pi$$

把球谐函数代入辐照度的计算等式（9），并展开，可得

$$\begin{aligned} E(\omega) \approx & \frac{1}{2\sqrt{\pi}}\hat{A}_0 L_0^0 + \frac{1}{2}\sqrt{\frac{3}{\pi}}\hat{A}_1\left(-L_1^1 x - L_1^{-1} y + L_1^0 z\right) \\ & + \frac{1}{2}\sqrt{\frac{15}{\pi}}\hat{A}_2\left(L_2^{-2} xy - L_2^{-1} yz - L_2^1 xz\right) + \frac{1}{4}\sqrt{\frac{5}{\pi}}\hat{A}_2 L_2^0 \cdot \left(3z^2 - 1\right) \\ & + \frac{1}{4}\sqrt{\frac{15}{\pi}}\hat{A}_2 L_2^2\left(x^2 - y^2\right) \end{aligned} \tag{11}$$

> 其中，$\frac{1}{2\sqrt{\pi}} \approx 0.282095$，$\frac{1}{2}\sqrt{\frac{3}{\pi}} \approx 0.488603$，$\frac{1}{2}\sqrt{\frac{15}{\pi}} \approx 1.09255$，$\frac{1}{4}\sqrt{\frac{15}{\pi}} \approx 0.546274$，$\frac{1}{4}\sqrt{\frac{5}{\pi}} \approx 0.315392$。

双向反射分布函数就是入射光辐照度与反射光辐射率的比值，漫反射光可以采用简单的Lambert模型，表示为

$$brdf(\omega_i, \omega_o) = \frac{1}{\pi}$$

那么，最终的光照计算公式就可以表示为

$$B(\omega_o) = \frac{1}{\pi}E(\omega_i)$$

我们把等式（10）和 $\hat{A}_0$、$\hat{A}_1$、$\hat{A}_2$ 的值代入上式，就得到环境贴图中漫反射光照的贡献值，表示为

$$+\frac{1}{8}\sqrt{15\pi}\left(L_2^{-2}xy - L_2^{-1}yz - L_2^1xz\right) + \frac{3}{16}\sqrt{5\pi}L_2^0z^2 \\ +\frac{1}{16}\sqrt{15\pi}L_2^2\left(x^2-y^2\right) \tag{12}$$

举个两个例子来解释如何利用离线生成的环境贴图的球谐系数来实现辐照度照明。

以虚幻引擎为例，参数IndirectLightingSHCoefficients表示的是光照函数的球谐系数，引擎实现中已经对这个参数除以PI，所以辐照度照明的实现如下所示：

```
/** The SH coefficients for the projection of a function that maps directions to scala
struct FThreeBandSHVector
{
    half4 V0;
    half4 V1;
    half V2;
};

half DotSH3(FThreeBandSHVector A,FThreeBandSHVector B)
{
    half Result = dot(A.V0, B.V0);
    Result += dot(A.V1, B.V1);
    Result += A.V2 * B.V2;
    return Result;
}

FThreeBandSHVector SHBasisFunction3(half3 InputVector)
{
    FThreeBandSHVector Result;
    // These are derived from simplifying SHBasisFunction in C++
    Result.V0.x = 0.282095f;
    Result.V0.y = -0.488603f * InputVector.y;
    Result.V0.z = 0.488603f * InputVector.z;
    Result.V0.w = -0.488603f * InputVector.x;

    half3 VectorSquared = InputVector * InputVector;
    Result.V1.x = 1.092548f * InputVector.x * InputVector.y;
    Result.V1.y = -1.092548f * InputVector.y * InputVector.z;
    Result.V1.z = 0.315392f * (3.0f * VectorSquared.z - 1.0f);
    Result.V1.w = -1.092548f * InputVector.x * InputVector.z;
    Result.V2 = 0.546274f * (VectorSquared.x - VectorSquared.y);

    return Result;
```

```
FThreeBandSHVector CalcDiffuseTransferSH3(half3 Normal,half Exponent)
{
    FThreeBandSHVector Result = SHBasisFunction3(Normal);

    // These formula are scaling factors for each SH band that convolve a SH with the
    // max(0,cos(theta))^Exponent
    half L0 =                   2 * PI / (1 + 1 * Exponent                          );
    half L1 =                   2 * PI / (2 + 1 * Exponent                          );
    half L2 = Exponent *        2 * PI / (3 + 4 * Exponent + Exponent * Exponent    );
    half L3 = (Exponent - 1) *  2 * PI / (8 + 6 * Exponent + Exponent * Exponent    );

    // Multiply the coefficients in each band with the appropriate band scaling factor
    Result.V0.x *= L0;
    Result.V0.yzw *= L1;
    Result.V1.xyzw *= L2;
    Result.V2 *= L2;

    return Result;
}

// Take the normal into account for opaque
FThreeBandSHVectorRGB PointIndirectLighting;
PointIndirectLighting.R.V0 = IndirectLightingCache.IndirectLightingSHCoefficients0[0];
PointIndirectLighting.R.V1 = IndirectLightingCache.IndirectLightingSHCoefficients1[0];
PointIndirectLighting.R.V2 = IndirectLightingCache.IndirectLightingSHCoefficients2[0];

PointIndirectLighting.G.V0 = IndirectLightingCache.IndirectLightingSHCoefficients0[1];
PointIndirectLighting.G.V1 = IndirectLightingCache.IndirectLightingSHCoefficients1[1];
PointIndirectLighting.G.V2 = IndirectLightingCache.IndirectLightingSHCoefficients2[1];

PointIndirectLighting.B.V0 = IndirectLightingCache.IndirectLightingSHCoefficients0[2];
PointIndirectLighting.B.V1 = IndirectLightingCache.IndirectLightingSHCoefficients1[2];
PointIndirectLighting.B.V2 = IndirectLightingCache.IndirectLightingSHCoefficients2[2];

FThreeBandSHVector DiffuseTransferSH = CalcDiffuseTransferSH3(MaterialParameters.World

// Compute diffuse lighting which takes the normal into account
half3 DiffuseGI = max(half3(0,0,0), DotSH3(PointIndirectLighting, DiffuseTransferSH));
```

再举另外一个例子，就是承接上一部分的实验，我们已经计算好了9个参数的球谐系数，我们可以采用如下所示的着色器代码来实现辐照度光照，得到的效果如图8所示，这里忽略了所有伽马

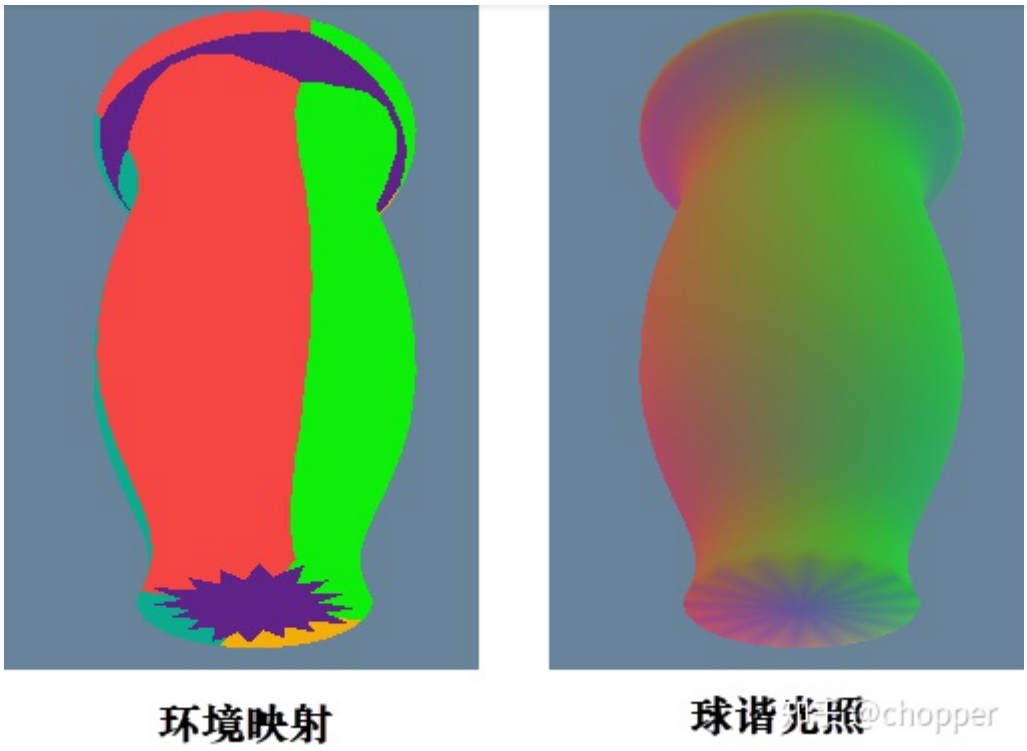


图8. 立方体映射与球谐辐照度照明

```
#version 100

precision mediump float;

mediump float calcSHIrradiance(mediump mat3 coef, mediump vec3 norm)
{
    mediump float l00 = coef[0][0];
    mediump float l1m1 = coef[1][0];
    mediump float l10 = coef[2][0];
    mediump float l11 = coef[0][1];
    mediump float l2m2 = coef[1][1];
    mediump float l2m1 = coef[2][1];
    mediump float l20 = coef[0][2];
    mediump float l21 = coef[1][2];
    mediump float l22 = coef[2][2];

    const mediump float PI = 3.1415926535897;
    const mediump float l0 = 1.0;
    const mediump float l1 = 2.0 / 3.0;
    const mediump float l2 = 1.0 / 4.0;
    const mediump float c0 = 0.282095;
    const mediump float c1 = 0.488603;
    const mediump float c2 = 1.09255;
    const mediump float c3 = 0.546274;
```

```
    mediump vec3 normSqr = norm * norm;

    mediump float irradiance =
        l00 * c0 * l0 +
        c1 * (-norm.y * l1m1 + norm.z * l10 - norm.x * l11) * l1 +
        c2 * (-norm.x * norm.y * l2m2 - norm.y * norm.z * l2m1 - norm.x * norm.z * l21
        c4 * (3.0 * normSqr.z - 1.0) * l20 * l2 +
        c3 * (normSqr.x - normSqr.y) * l22 * l2;
    return irradiance;
}

uniform mediump mat3 sh_r;
uniform mediump mat3 sh_g;
uniform mediump mat3 sh_b;
uniform samplerCube u_cube;
varying mediump vec3 v_normal;
uniform bool use_cube;

void main()
{
    vec3 ambient = vec3(1.0);
    if (use_cube)
    {
        ambient = textureCube(u_cube, v_normal).xyz;
    }
    else
    {
        ambient = vec3(calcSHIrradiance(sh_r, v_normal), calcSHIrradiance(sh_g, v_norm
    }

    gl_FragColor = vec4(ambient, 1.0);
}
```

此外，你也可以在shadertoy上实现一个简单的基于球谐函数的辐照度照明效果，如图9所示，实现的源码如下所示。

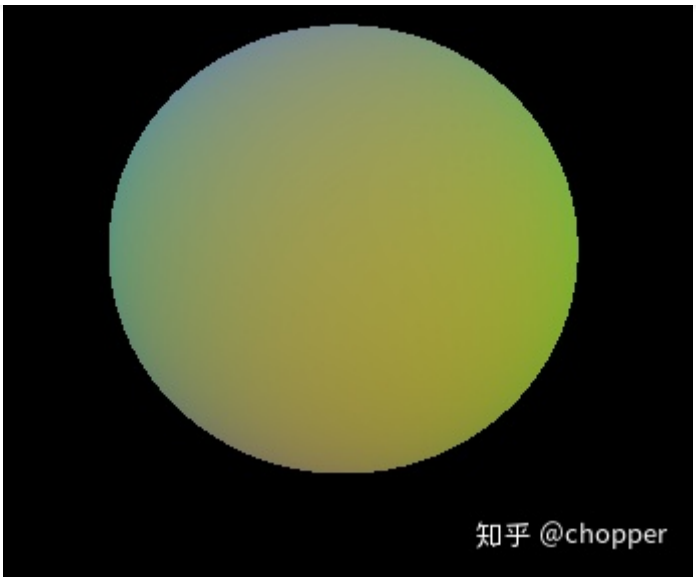


图9. 基于球谐函数的辐照度照明

```
// by tkstar.
// Lighting is done by Spherical Harmonics:
// This one is a cheap variant presented in 2001 by Ravi Ramamoorthi
struct SHCoefficients {
    vec3 l00, l1m1, l10, l11, l2m2, l2m1, l20, l21, l22;
};

const SHCoefficients stpeter = SHCoefficients(
    vec3( 1.649656, 1.818793, 1.422597),
    vec3( -0.010007, -0.210143, -0.360245),
    vec3( 0.003336, 0.226821, -0.443635),
    vec3( 0.006671, -0.346903, 0.196801),
    vec3( 0.000000, -0.000000, 0.000000),
    vec3( -0.000000, -0.000000, -0.000000),
    vec3( -0.901174, 0.635535, -0.212797),
    vec3( 0.000000, -0.000000, -0.000000),
    vec3( 0.719835, 0.279523, -0.764361)
);

vec3 calcSHIrradiance(vec3 norm, float scale){
    const float PI = 3.1415926535897;
    const float l0 = 1.0;
    const float l1 = 2.0 / 3.0;
    const float l2 = 1.0 / 4.0;
    const float c0 = 0.282095;
    const float c1 = 0.488603;
    const float c2 = 1.09255;
    const float c3 = 0.546274;
```

```
    const SHCoefficients c = stpeter;
    vec3 normSqr = norm * norm;

    vec3 irradiance =
        c.l00 * c0 * l0 +
        c1 * (-norm.y * c.l1m1 + norm.z * c.l10 - norm.x * c.l11) * l1 +
        c2 * (-norm.x * norm.y * c.l2m2 - norm.y * norm.z * c.l2m1 - norm.x * norm.z *
        c4 * (3.0 * normSqr.z - 1.0) * c.l20 * l2 +
        c3 * (normSqr.x - normSqr.y) * c.l22 * l2;
    return irradiance * scale;
}

vec3 spherePos = vec3(0.0, 1.0, 1.5);
float sphereRadius = 2.5;

float raytraceSphere(in vec3 ro, in vec3 rd, float tmin, float tmax, float r) {
    vec3 ce = ro - spherePos;
    float b = dot(rd, ce);
    float c = dot(ce, ce) - r * r;
    float t = b * b - c;
    if (t > tmin) {
        t = -b - sqrt(t);
        if (t < tmax)
            return t;
        }
    return -1.0;
}

void mainImage( out vec4 fragColor, in vec2 fragCoord )
{
    vec2 p = (2.0 * fragCoord.xy - iResolution.xy) / iResolution.y;

    float time = iTime;
    vec3 eye = vec3(0.0, 3.0, 5.0);
    vec2 rot = 6.2831 * (vec2(0.6 + time * 0.05, sin(time * 0.1) * 0.06) + vec2(1.0, 0
    eye.yz = cos(rot.y) * eye.yz + sin(rot.y) * eye.zy * vec2(-1.0, 1.0);
    eye.xz = cos(rot.x) * eye.xz + sin(rot.x) * eye.zx * vec2(1.0, -1.0);

    vec3 ro = eye;
    vec3 ta = vec3(0.0, 1.0, 0.0);

    vec3 cw = normalize(ta - eye);
    vec3 cu = normalize(cross(vec3(0.0, 1.0, 0.0), cw));
```

```
    vec3 rd = cam * normalize(vec3(p.xy, 1.0));

    vec3 col = vec3(0, 0, 0);
    float tmin = 0.1;
    float tmax = 50.0;

    // raytrace the sphere
    float tsph = raytraceSphere(ro, rd, tmin, tmax, sphereRadius);
    if (tsph > tmin) {
        vec3 spos = ro + rd * tsph;
        vec3 nor = normalize(spos - spherePos);
        col = calcSHIrradiance(nor, 2.0);
    }

    fragColor = vec4(col, 1.0);
}
```

参考

[1] Ravi Ramamoorthi, and Pat Hanrahan. "On the relationship between radiance and irradiance: determining the illumination from images of a convex Lambertian object." JOSA A 18.10: 2448-2459, 2001.

[2] Ravi Ramamoorthi, and Pat Hanrahan. "An efficient representation for irradiance environment maps." Proceedings of the 28th annual conference on Computer graphics and interactive techniques. 2001.

[3] Gary King. "Real-time computation of dynamic irradiance environment maps." GPU Gems 2, 167-176, 2005.

[4] github, justinmeiners/spherical-harmonics

[5] Rory. CUBEMAP TEXEL SOLID ANGLE

[6] Peter-Pike Sloan. "Stupid spherical harmonics (sh) tricks." Game developers conference. Vol. 9. 2008.

编辑于 05-26

赞赏

1 人已赞赏

实时渲染 虚幻引擎 全局光照

文章被以下专栏收录

chopper的图形渲染
记录游戏图形渲染相关的文章

推荐阅读

大气散射：4.预计算之多重散射

多重散射我们认为是前一阶散射/地面反射完成后又进行的下一次散射。 跟single scattering类似，我们需要计算整条视线上的积分。但区别在于我们在计算每一点处散射到视线方向的光的时候，需...

Snafloda-豆娘

【球谐光照】(三)光源和光照计算

在前两篇文章中我们分别讲了球谐函数的生成以及如何利用球谐函数去近似一个光源，得到低频的球谐光源： xiaomengge：【球谐光照】(一)从拉普拉斯方程到球谐函数 xiaomengge：【球谐光照】(二...

XMGXMG

实

14

19 条评论

切换为时间排序

写下你的评论...

张晓宇 2020-11-27

生成辐照度贴图那个伪代码没写对啊。。

1

大佬，UE4的实现里面，CalcDiffuseTransferSH3这个函数里面的实现没看懂，和上面的推导感觉没啥关系啊

赞

chopper (作者) 回复 Superlee 08-06

没看懂就瞪大眼再看！

赞

pinkman 08-01

讲的最清楚的一篇

赞

jzh 07-29

您好，我阅读了您的关于球协函数光照的文章，请问您有相应的工程文件吗？能否分享一下呢？谢谢！

赞

戴斯尔 03-06

想问一下这里的球谐函数坐标系是需要和法向信息坐标系一致吗🙏

赞

chopper (作者) 回复 戴斯尔 03-06

我理解的球谐函数并不存在坐标系的概念，你就把它理解为三维数据即可。

1

戴斯尔 回复 chopper (作者) 03-06

非常感谢大佬回复🙏那如果法向信息已知，其对应的空间坐标系也是已知，在使用通过一组球谐系数进行表示的光照进行渲染时，怎么确定哪个半球面是对象正面来光，哪个半球面是对象背面来光呢🙏

赞

展开其他 2 条回复

一天到头困来的 01-29

请问下，为什么我看An Efficient Representation for Irradiance Environment Maps 这篇论文里，球谐函数Y00-Y22都是非负的，而文章里面有几项是负的。但是从最后B(w)的公式来看和StupidSH这篇论文里最后提供的代码是一样的，不知道原因是什么。。

赞

的，并没有验证过是否是不对的。现在工程应用中，都有这一项，就以存在负数的为准了。

赞

一天到头困来的 回复 chopper (作者) 03-06

好的，谢谢。。

赞

MiaoMiaoM 2020-12-05

想请教大佬一个问题，计算漫反射需要的环境信息是给定一个方向，得到以该方向为轴心的半球上的总辐照度，开头的辐照度环境贴图正是提供了这个信息。那么为什么不直接用球谐来记录辐照度信息呢？为什么是使用球谐记录辐射度，还要引入传递函数这些东西呢？

赞

lok666 回复 MiaoMiaoM 01-24

记录辐照度信息就是的话就是传统IBL的做法，要存个cubemap，相较于只记录几个sh系数来说显存，内存开销都太大了。划分成传输函数是为了应对光源函数可能实时变化的情况（比如旋转）。

1

大画渣 2020-08-04

聚聚牛逼

赞

Shadowell 2020-07-11

聚聚牛逼

赞

chopper (作者) 回复 Shadowell 2020-07-13

谁是聚聚?

赞

Shadowell 回复 chopper (作者) 2020-07-13

聚聚就是巨佬的意思

赞