Glint Rendering

141160033 廖祥森

1. 原理

Glint 模型以 microfacet 模型为基础，但由于 glint 中的法线分布为离散的分布，因此替换 BRDF 中的法线分布函数 D(x, w_h)函数为 D(A, Ω_h)。其中 A 代表了光线与物体交点的周围一小块区域；Ω_h 代表了以相应的出射光线为中线，γ为圆锥角的相应圆锥。替换之后的 BRDF 为：

$$\frac{(\boldsymbol{\omega}_i \cdot \boldsymbol{\omega}_h)\, F(\boldsymbol{\omega}_i \cdot \boldsymbol{\omega}_h)\, \widehat{D}(A, \Omega_h)\, G(\boldsymbol{\omega}_i, \boldsymbol{\omega}_o, \boldsymbol{\omega}_h)}{a(A)\, \sigma(\Omega_o)\, (\boldsymbol{\omega}_i \cdot \mathbf{n_x})(\boldsymbol{\omega}_o \cdot \mathbf{n_x})}$$

又由于其中的 D 函数不能给出参数化的公式，因此通过 counting 的方法来计算这个值，简单的来说就是事先生成好微表面的分布，对于给定的 A 和Ω_h，判断有多少微表面在空间位置上落在 A 中，在法线方向上落在Ω_h中。D 最后就表示为落在 A 中的微表面比例乘以落在Ω_h中的微表面比例。

又由于事先生成好微表面的分布太占内存空间，并且事实上我们其实并不需要每个微表面的位置及发现方向，我们只需要直到给定一块区域，有多少块满足条件的微表面。因此，我们将整个材质平面和法线分布半球面进行四叉树划分，我们只需给出若干个(x, y, z, w)(x + y + z + w = 1)四元组，将整个区域不断划分，直到我们可以得出 A 和Ω_h中有多少满足条件的微表面。

对于具体如何计算 A 和Ω_h中满足条件的微表面数，我们运用如下算法 1：

```
function D̂(D, N, A, ω_i, ω_o, γ)
    query ← A × ConicSection(ω_i, ω_o, γ)
    queue ← { root node of tree }
    count ← 0
    while queue ≠ ∅ do
        node ← queue.pop()
        if node ∩ query = ∅ or |node| = 0 then
            pass
        else if node ⊆ query then
            count ← count + |node|
        else if error criterion (10) satisfied then
            count ← count + |node| ·
              (node ∩ query).vol() / node.vol()
        else
            for c in node.split() do
                queue.push(c)
    return count / N
```

2. 实现方法

2.1. 生成(x,y,z,w)四元组

为保证微表面的分布不会太过于不均匀，因此在生成四元组时，要保证 x、y、z、w 之间的方差不会过大，随着整个区域分划次数的增加，所允许的最大方差逐渐增加。假设 n 为分划次数，所允许的方差要小于 $1.0 - 0.96*1.1^{-n}$

2.2. 计算 A 的范围

首先根据交点 p 以及入射光线 w_i，计算出光线的源点 o，以及该光线与成像平面的交点 p_film，根据 p_film 计算出其所处像素的四个交点，计算得出相应的四条光线 ray_diff[4]。

根据交点 p 所处的三角面片 Tri 的法相，计算出材质平面，求材质平面与 ray_diff[4]的交点 intersect_diff[4]。再根据 Tri 的三个顶点的空间坐标与材质坐标的关系，计算出 intersect_diff[4]的相应材质坐标 tex_diff[4]。对 tex_diff[4]求轴对齐的二维包围盒 tex_box，该包围盒就代表了 A 的范围。

```
Vector ray_d = its.toWorld(-its.wi), d_diff[4];
Point ray_o = its.p - its.t * ray_d, p_film = ray_o + camera->getNearClip() * ray_d,
        sample_diff[4], intersect_diff[4];
Point2 tex_diff[4];
Ray ray_diff[4];
camera->get_sample_differential(p_film, sample_diff);
TexPlane plane = TexPlane(its.p, Normal(its.toWorld(Vector(0, 0, 1.f))));
// Sample texture coord for intersect diff
plane.add_tri(mesh->getVertexPositions(), mesh->getVertexTexcoords(), tri.idx);
for(int i = 0; i < 4; i++)
{
    float t;
    d_diff[i] = (sample_diff[i] - ray_o) / camera->getNearClip();
    ray_diff[i] = Ray(ray_o, d_diff[i], 0.f);
    if(plane.intersect(ray_diff[i], t))
        intersect_diff[i] = (ray_diff[i])(t);
    else {
        return Spectrum(0.5f);
    }
    plane.sample_tex_coord(intersect_diff[i], tex_diff[i]);
}
```

2.3. 计算 A 中的微表面比例

以 tex_box 为 query，(0,1)的正方形材质空间为 node，由于两者都是轴对齐的，我们可以很容易的判定 query 与 node 的相对位置关系，运用算法 1，我们就可以得到 A 中的微表面比例。算法的实现如下：

```
while(!queue.empty())
{
    node = queue.back();
    queue.pop_back();
    if(!node.first.overlaps(query) || node.second <= 0) {
        // Pass
    }
    else if(query.contains(node.first)) {
        count += node.second;
    }
    else {
        p = count / SPATIAL_SAMPLE_NUM;
        if(sqrt(node.second * p * (1 - p)) < EPSILON_COUNT * count) {
            clip_box = node.first;
            clip_box.clip(query);
            count += (uint32_t)(node.second * clip_box.getSurfaceArea() / node.first.getSurfaceArea());
        }
        else {
            assert(split_times < SET_P_SIZE);
            Point2 center = node.first.getCenter(), min = node.first.min, max = node.first.max;
            queue.push_back(std::make_pair(
                    AABB2(min, center),
                    (uint32_t)(node.second * set_p[split_times].x)));
            queue.push_back(std::make_pair(
                    AABB2(Point2(center.x, min.y), Point2(max.x, center.y)),
                    (uint32_t)(node.second * set_p[split_times].y)));
            queue.push_back(std::make_pair(
                    AABB2(center, max),
                    (uint32_t)(node.second * set_p[split_times].z)));
            queue.push_back(std::make_pair(
                    AABB2(Point2(min.x, center.y), Point2(center.x, max.y)),
                    (uint32_t)(node.second * set_p[split_times].w)));
            split_times++;
```

其中 set_p 存储了(x,y,z,w)四元组，node.first 代表了相应的轴对齐包围盒，node.second 代表了 node 中的微表面数。

2.4. 计算Ω_h中的微表面比例

Ω_h代表了出射光线的分布范围，我们将其转换为相应的半角 m 的分布范围，其中 m = normalize(w_i+w_o)，以该区域作为 query：

$$\|\mathbf{m}\| = 1 \text{ and } \mathbf{m}^T\mathbf{C}\mathbf{m} \le 0, \text{ where } \mathbf{C} = \mathbf{Q}\mathbf{\Lambda}\mathbf{Q}^T \text{ and}$$

$$\mathbf{Q} = \begin{bmatrix} | & | & | \\ \hat{\mathbf{x}} & \hat{\mathbf{y}} & \hat{\mathbf{z}} \\ | & | & | \end{bmatrix}, \quad \mathbf{\Lambda} = \begin{bmatrix} \lambda_1 & & \\ & \lambda_2 & \\ & & -1 \end{bmatrix},$$

$$\lambda_1 = \frac{\boldsymbol{\omega}_i \cdot \boldsymbol{\omega}_o + \cos\gamma}{1 - \cos\gamma}, \quad \lambda_2 = \cot^2\frac{\gamma}{2}.$$

Node 的初始值为整个半球面，首先将其划分为四等份，之后的每次划分都取球面三角各边的中点连线，划分成相应的四份。

对于如何求 query 与 node 的位置关系，分为两步。

首先判断两者的边线是否有交点，对于如何判断 node 的一条边 ab 与 query 是否相交，运用如下公式：

$$\mathbf{d}^T\mathbf{C}\mathbf{d}t^2 + 2\mathbf{d}^T\mathbf{C}\mathbf{c}t + \mathbf{c}^T\mathbf{C}\mathbf{c} = 0$$

其中 d=a-b,c=a+b,C 为 query 的相关参数，当 t 有处于-1 到 1 之间的根时，ab 与 query 相交，具体的计算过程如下：

```
Point3 tri[3] = {Spherical2Cartesian(_tri[0]), Spherical2Cartesian(_tri[1]), Spherical2Cartesian(_tri[2])};
Vector c, d;
Float param_a, param_b, param_c, axis;
for(int i = 0; i < 3; i++)
{
    c = tri[i] + tri[(i + 1) % 3] - Point3(0.f, 0.f, 0.f);
    d = tri[i] - tri[(i + 1) % 3];
    param_a = dot(C.preMult(d), d);
    param_b = 2 * dot(C.preMult(d), c);
    param_c = dot(C.preMult(c), c);
    if((param_a - param_b + param_c) * (param_a + param_b + param_c) < 0.f)
        return true;
    else
    {
        axis = - param_b / (2 * param_a);
        if(axis < -1.f || axis > 2.f || param_a * param_c - param_b * param_b * 0.25f >= 0.f)
            continue;
        else
            return true;
    }
}
return false;
```

当 node 与 query 的边线不相交时，若 node 的三个顶点都落在 query 中，则 node ⊆ query，又由于 query ⊆ node 不可能存在，因此可以得出两者之间的位置关系，运用算法 1 就可计算出Ω_h中的微表面比例。

3. 效果