vcl final project report

Xinzhe Wu

2026.1.10

1 选题:C Rendering - 5. Path Tracing 基于渲染方程的全局真实感渲染

基于 [https://www.kevinbeason.com/smallpt/]smallpt 实现的 path tracing 算法(代码在 src/VCX/Labs/Final_Project) 相比于 lab3 中的 whitted-style ray tracing 算法，其主要优化在于 1. 对于点光源，看成球状光源利用均匀分布多次采样 2. 对于反射过程进行蒙特卡洛采样 3. 采用 Russian roulette ，以一定概率提前终止 trace，最终可以实现更加真实的渲染结果以及更柔和的阴影边缘

2 算法实现

```
glm::vec3 PathTrace(const RayIntersector & intersector, Ray ray, int maxDepth, int pixel_samples, int seed) {
        glm::vec3 color(0.0f);
        glm::vec3 throughput(1.0f);
        const float color_rate = 2.5f;
        const float ambient_rate = 4.5f;
        const int samples    = 8;
        const float light_radius = 10.0f;

        std::mt19937                     gen(seed);
        std::uniform_real_distribution<float> dis(0.0f, 1.0f); // 初始化结果，参数以及随机数生成器

        for (int depth = 0; depth < maxDepth; ++depth) { // 递归深度
            auto rayHit = intersector.IntersectRay(ray);
            if (! rayHit.IntersectState) {
                color += throughput * glm::vec3(0.0f, 0.0f, 0.0f);
                break; // 未相交则中止
            }

            const glm::vec3 pos        = rayHit.IntersectPosition;
            const glm::vec3 nl         = glm::normalize(rayHit.IntersectNormal);
            const glm::vec3 kd         = glm::vec3(rayHit.IntersectAlbedo);
            const glm::vec3 ks         = glm::vec3(rayHit.IntersectMetaSpec);
            const float   shininess     = rayHit.IntersectMetaSpec.w * 256.0f;
            const glm::vec3 n          = glm::dot(ray.Direction, nl) < 0 ? nl : -nl;
            const float   specular_weight = glm::length(ks);
            const float   diffuse_weight = glm::length(kd);
            for (const auto & light : intersector.InternalScene->Lights) {
                glm::vec3 light_total = glm::vec3(0.0f);
                for (int s = 0; s < samples; s++) { //每条光线采样多次
                    glm::vec3 l;
                    float    attenuation   = 1.0f;
                    glm::vec3 sampledLightPos = light.Position;
                    if (light.Type == Engine::LightType::Point) { //点光源从单位球中采样offset
                        float    u1  = dis(gen);
                        float    u2  = dis(gen);
                        float    z   = 1.0f - 2.0f * u1;
                        float    r_xy = sqrt(1.0f - z * z);
                        float    phi  = 2.0f * glm::pi<float>() * u2;
                        glm::vec3 offset(r_xy * cos(phi), r_xy * sin(phi), z);
                        offset *= light_radius;
                        sampledLightPos = light.Position + offset;
                        l            = sampledLightPos - pos;
                        attenuation  = 1.0f / glm::dot(l, l);
```

```cpp
                } else if (light.Type == Engine::LightType::Directional) {
                    l              = light.Direction;
                    sampledLightPos = pos + l * 1000.0f;
                }
                l = glm::normalize(l);

                // Shadow ray
                Ray shadowRay(pos + n * EPS1, l); //计算shadow ray和 color 同 ray tracing 一样
                auto shadowHit = intersector.IntersectRay(shadowRay);
                if (! shadowHit.IntersectState || glm::distance(pos, shadowHit.IntersectPosition) > glm::distance(pos, sampledLightPos)) {
                    float     cosTheta      = glm::max(glm::dot(n, l), 0.0f);
                    glm::vec3 diffuse_color = kd * cosTheta * attenuation * light.Intensity * color_rate;
                    glm::vec3 norm_eye    = glm::normalize(-ray.Direction);
                    glm::vec3 norm_half   = glm::normalize(l + norm_eye);
                    float     specular_angle = glm::max(glm::dot(n, norm_half), 0.0f);
                    glm::vec3 specular_color = ks * pow(specular_angle, shininess) * attenuation * light.Intensity * color_rate;
                    light_total += diffuse_color + specular_color;
                }
            }
            light_total /= float(samples);
            color += throughput * light_total;
        }

        // 加载环境光
        color += throughput * kd * intersector.InternalScene->AmbientIntensity * ambient_rate;

        // 蒙特卡洛采样利用均匀分布模拟以一定概率发生镜面反射，否则发生漫反射，漫反射从以法向为中心的半球进行余弦加权采样
        glm::vec3 newDir;
        glm::vec3 bsdf_contribution;
        float     r_brdf = dis(gen);
        if (r_brdf < specular_weight && specular_weight > 0.01f) {//镜面反射
            newDir            = glm::reflect(ray.Direction, n);
            newDir            = glm::normalize(newDir);
            bsdf_contribution = ks;

            throughput *= bsdf_contribution / specular_weight;
        } else { // 漫反射半球余弦加权采样
            float r1       = dis(gen);
            float r2       = dis(gen);
            float phi      = 2.0f * glm::pi<float>() * r1;
            float cosTheta = sqrt(r2);
            float sinTheta = sqrt(1.0f - r2);

            glm::vec3 tangent = glm::normalize(glm::cross(n, glm::vec3(0, 1, 0)));
            if (glm::length(tangent) < 0.1f) tangent = glm::normalize(glm::cross(n, glm::vec3(1, 0, 0)));
            glm::vec3 bitangent = glm::cross(n, tangent);

            newDir            = cosTheta * n + sinTheta * (cos(phi) * tangent + sin(phi) * bitangent);
            newDir            = glm::normalize(newDir);
            bsdf_contribution = kd;

            throughput *= bsdf_contribution / (1.0f - specular_weight);
        }

        ray = Ray(pos + n * EPS1, newDir);

        // Russian roulette 以一定的概率提前停止
        float p = glm::max(throughput.r, glm::max(throughput.g, throughput.b));
        p       = glm::max(p, 0.15f);
        if (depth > 3 && dis(gen) > p) break;
        if (depth > 3) throughput /= p;
    }
    return color;
}
```

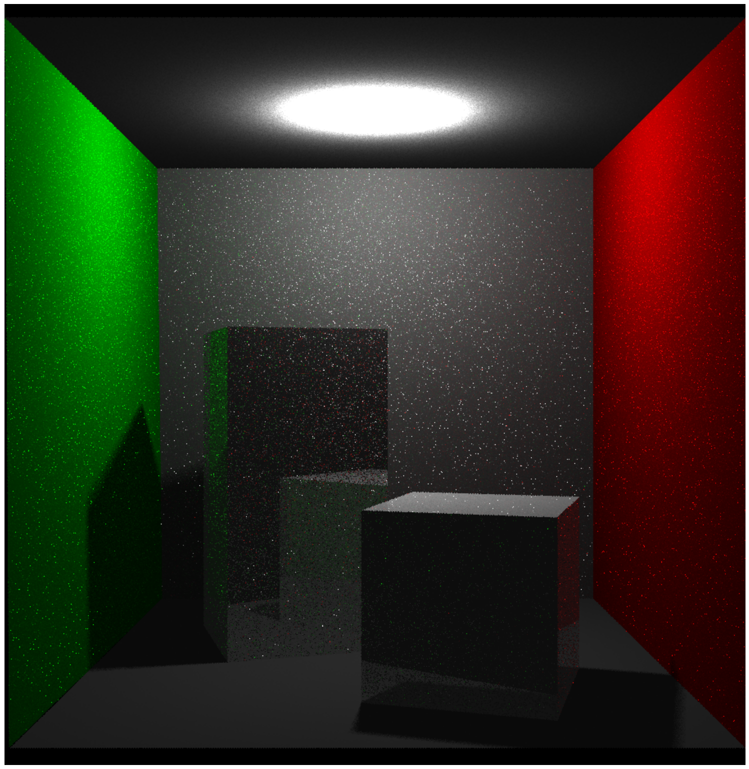


图 1: pathtracing 的噪点（sample=1，depth=3）

3 结果分析

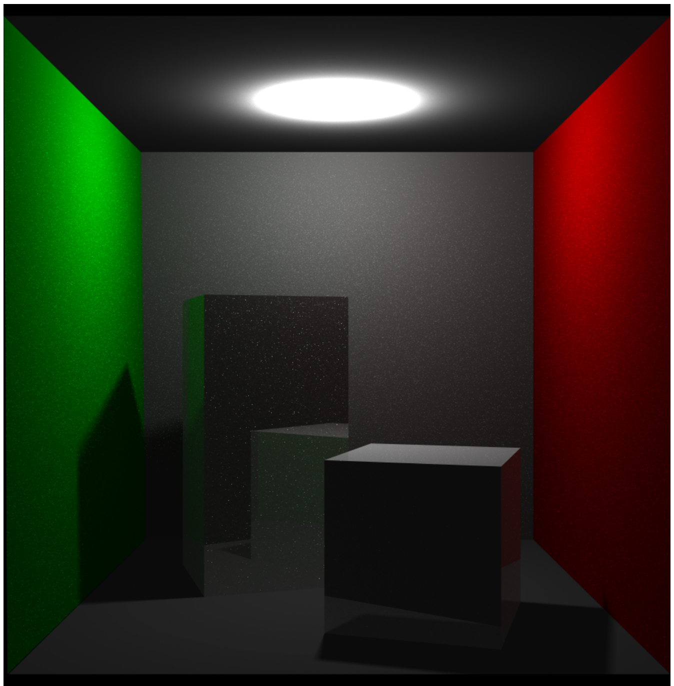
1. 使用我实现的 path tracing 算法相比于 whitted-style 渲染真实感更明显，相比于 whitted 的锐利阴影阴影更加柔和

2. 在 sample pixels 较小时，图像会有较多的噪点 (图一)，这是因为采样数太少导致导致算法中的随机过程扰动明显（均匀分布可进一步优化）

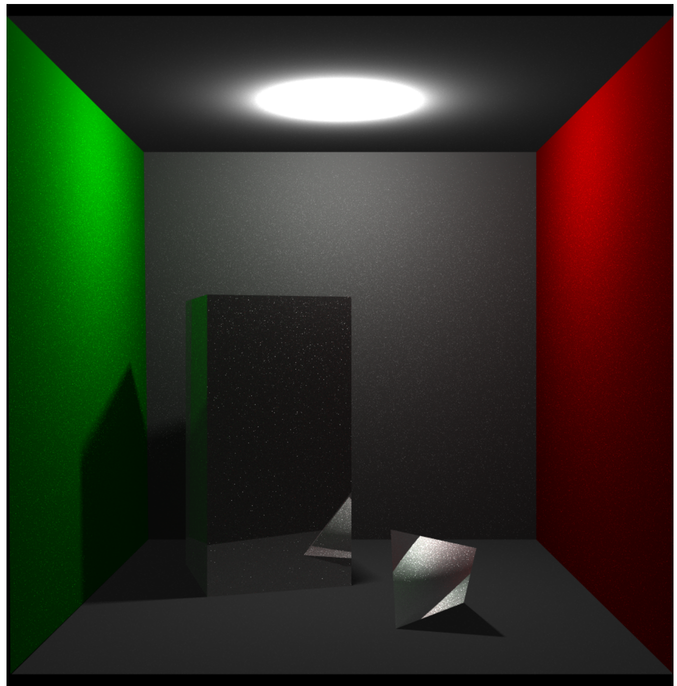
3.path tracing 得到的结果相比于 whitted-style 会暗一些，这可能是因为，漫反射余弦采样导致的能量各方向损耗，Russian roulette 的提前中止

在最后一页附上 path tracing（sample=70,depth=12）和 whitted-style ray tracing 的效果对比

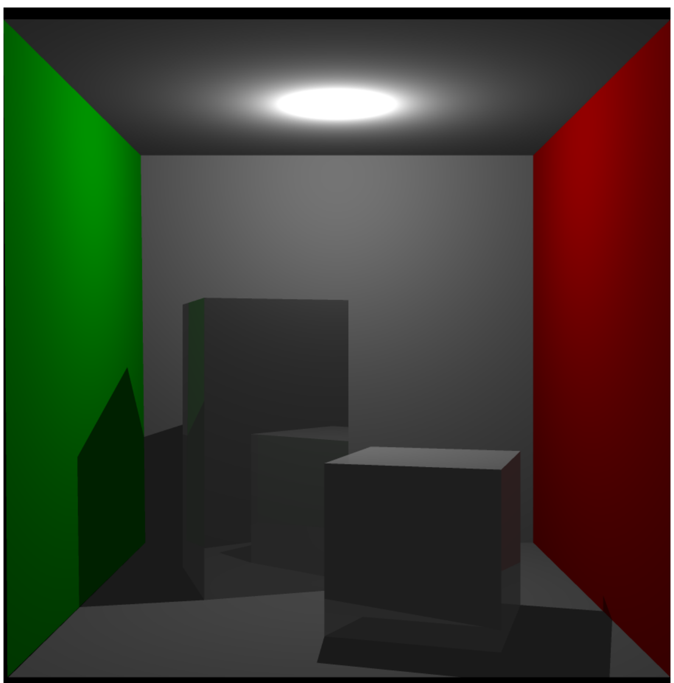
4 结果对比展示



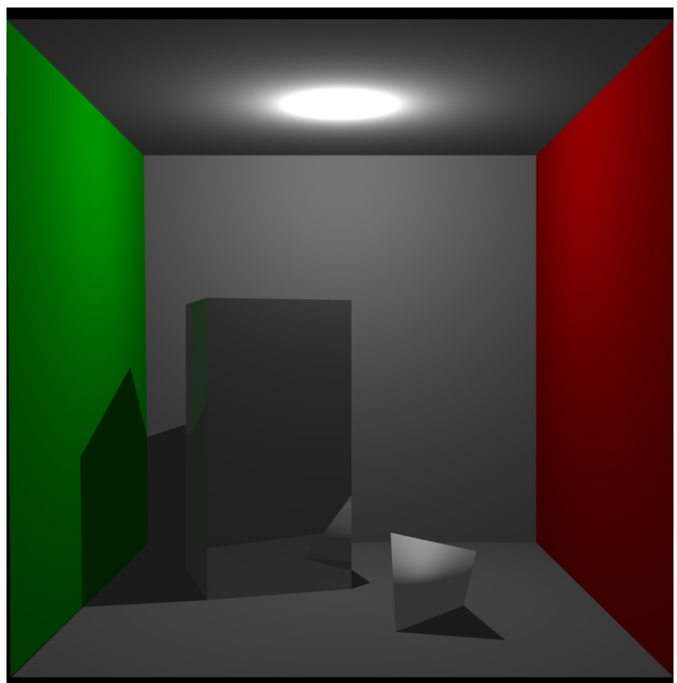
(a) pathtracing



(b) pathtracing



(c) whitted-style



(d) whitted-style

图 2: pathtracing 和 whitted-style 对比