高等计算机图形学大作业

任毅 计科02 2020011624

一、渲染参数

渲染平台	GPU
平台型号	NVIDIA RTX 3070
操作系统	Windows 10
代码语言	GLSL, C++

二、功能及特性

（一）多种材质

1. 漫反射（Diffuse / Lambertian）

BRDF为常数的材质，即

$$f(\omega_i, \omega_o) = c/\pi,$$

其中 ω_i 为入射方向，ω_o为出射方向，c 为固有色（下同）。渲染结果参见Cornell Box。

代码位于github仓库中`/srcs/spark/shaders/path_tracing.rgen/SampleRay1(...)`，核心代码：

```
wo = -wi;
wi = ImportanceSamplingLamberrian(n, t);
Fr = material.diffuse *
              vec3(texture(texture_samplers[material.diffuse_texture_id],
                    hit_record.tex_coord));
throughput *= Fr / prob;

radiance += throughput * SampleEmission(p, n) * INV_PI;
```

2. 自发光（Emissive）

余弦发光表面，radiance为

$$Li(w_o) = e,$$

其中 e 为发光强度，即发光颜色乘以强度。渲染结果参见Cornell Box。

以diffuse的光源采样为例，代码位于github仓库中`/srcs/spark/shaders/path_tracing.rgen`核心代码：

```
vec3 SampleEmission(vec3 origin, vec3 origin_normal) {
  vec3 energy = vec3(0.0);
  uint emission_id = RandomInt(guo.num_emissions);
  uint emission_entity_id = emissions[emission_id];
  ObjectInfo object_info = object_infos[emission_entity_id];
  uint face_id = RandomInt(object_info.num_faces);
  Vertex v0 = GetVertex(
      object_info.vertex_offset +
      indices[object_info.index_offset + face_id * 3 + 0]);
  Vertex v1 = GetVertex(
```

```
      object_info.vertex_offset +
      indices[object_info.index_offset + face_id * 3 + 1]);
  Vertex v2 = GetVertex(
      object_info.vertex_offset +
      indices[object_info.index_offset + face_id * 3 + 2]);
  mat3 object_to_world = mat3(entity_objects[emission_entity_id].model);
  mat3 positions = mat3(entity_objects[emission_entity_id].model *
                        mat3x4(vec4(v0.position, 1.0), vec4(v1.position, 1.0),
                               vec4(v2.position, 1.0)));
  float angle;
  vec3 wi;
  SampleSolidAngle(origin, positions, angle, wi);
  HitRecord save_hit_record = hit_record;
  if (TraceRay(origin, wi)) {
    if (hit_record.hit_entity_id == emission_entity_id &&
        ray_payload.primitive_id == face_id) {
      energy = materials[emission_entity_id].emission *
               materials[emission_entity_id].emission_strength *
               max(dot(origin_normal, wi), 0.0) * angle *
               object_info.num_faces * guo.num_emissions;
    }
  }
  hit_record = save_hit_record;
  return energy;
}
```

3. 镜面反射（Specular）

含有垂直反照率（F_0）、粗糙度（roughness）参数的金属反射，BRDF为

$$f(l,v)=\frac{F(l,h)G(l,v)D(h)}{4(n\cdot l)(n\cdot v)},$$

其中 l 为入射方向，v为出射方向，h为微观法向量，n为宏观法向量。

F为菲涅尔项，近似为以下函数，

$$F(l,h)=F_0+(1-F_0)(1-l\cdot h)^5.$$

G为几何遮蔽项，近似为以下函数，

$$\begin{aligned}G(l,v)&=\frac{0.5}{\Lambda(l)+\Lambda(v)},\\ \Lambda(l)&=(n\cdot v)((n\cdot l)(1-\alpha)+\alpha),\\ \Lambda(v)&=(n\cdot l)((n\cdot v)(1-\alpha)+\alpha),\\ \alpha&=roughness^2.\end{aligned}$$

D为微观法线分布函数，使用了GGX（Trowbridge-Reitz）分布，

$$\begin{aligned}D(h)&=\frac{\alpha^2}{\pi((n\cdot h)^2(\alpha^2-1)+1)^2},\\ \alpha&=roughness^2.\end{aligned}$$

渲染结果参见Cornell Box Specular、Cornell Box Rough Specular。

代码位于github仓库中`/srcs/spark/shaders/path_tracing.rgen/SampleRay1(...)`，核心代码：

```
wo = -wi;
float alpha = material.roughness * material.roughness;
vec3 wh = ImportanceSamplingSpecular(n, t, alpha);
wi = 2.0 * wh * dot(wh, wo) - wo;
vec3 F0 = material.specular *
    vec3(texture(texture_samplers[material.specular_texture_id],
                 hit_record.tex_coord));
float oh = max(dot(wo, wh),0.0);
vec3 F = F0 + (1.0 - F0) * pow5(1.0 - oh);
float no = max(dot(n, wo), 0.0);
float ni = max(dot(n, wi), 0.0);
float nh = max(dot(n, wh), 0.0);
float Vis_SmithVL = no * (ni * (1.0 - alpha) + alpha) +
    ni * (no * (1.0 - alpha) + alpha);
if (Vis_SmithVL <= 0.0 || nh == 0.0) {
    Fr = vec3(0.0);
}
else {
    Fr = F * clamp(2.0 * oh * ni / nh / Vis_SmithVL, 0.0, 1.0);
}
throughput *= Fr / prob;

radiance += throughput * materials[hit_record.hit_entity_id].emission;
```

4. 透射（Transmissive）

含有双面垂直透射率（T）、折射率（ior, η）、粗糙度（roughness）参数的镜面反射与透射，BRDF为，

反射项：

$$f(l,v)=\frac{F(l,h)G(l,v)D(h)}{4(n\cdot l)(n\cdot v)},$$

折射项：

$$f(l,v)=\frac{(1-F(l,h))G(l,v)D(h)\left|l\cdot h\right|\left|v\cdot h\right|}{4(n\cdot l)(n\cdot v)\left\|l+\eta v\right\|^2},\eta=\frac{\eta_l}{\eta_v}.$$

F为菲涅尔项，

$$F(l,h)=\frac{R_s+R_p}{2},$$

$$R_s=\left|\frac{\eta_l\cos\theta_l-\eta_v\cos\theta_v}{\eta_l\cos\theta_l+\eta_v\cos\theta_v}\right|^2,R_p=\left|\frac{\eta_l\cos\theta_v-\eta_v\cos\theta_l}{\eta_l\cos\theta_v+\eta_v\cos\theta_l}\right|^2,$$

$$\cos\theta_l=l\cdot h,\cos\theta_v=\sqrt{\max\left\{0,1-\frac{\eta_l^2}{\eta_v^2}\sqrt{1-\cos^2\theta_l}\right\}}.$$

G与D与镜面反射相同。渲染结果参见Cornell Box Transmissive、Cornell Box with Specular Lucy and Transmissive Bunny。

代码位于github仓库中`/srcs/spark/shaders/path_tracing.rgen/SampleRay1(...)`，核心代码：

```
wo = -wi;
```

```
float alpha = material.roughness * material.roughness;
vec3 wh = ImportanceSamplingSpecular(n, t, alpha);
float oh = max(dot(wo, wh), 0.0);
float F = FrDielectric(oh, 1.0, material.ior, hit_record.front_face);
vec3 T = sqrt(material.transmittance);
Fr = vec3(0.0);
if (RandomFloat() < F) {
    wi = 2.0 * wh * dot(wh, wo) - wo;
    float no = max(dot(n, wo), 0.0);
    float ni = max(dot(n, wi), 0.0);
    float nh = max(dot(n, wh), 0.0);
    float Vis_SmithVL = no * (ni * (1.0 - alpha) + alpha) +
        ni * (no * (1.0 - alpha) + alpha);
    if (Vis_SmithVL > 0.0 && nh > 0.0) {
        Fr = T * clamp((1.0, 2.0 * oh * ni / nh / Vis_SmithVL), 0.0, 1.0);
    }
} else {
    float eta = hit_record.front_face ? (1.0 / material.ior) : (material.ior /
1.0);
        if (Refract(wo, wh, eta, wi)) {
            float ih = max(dot(wi, -wh), 0.0);
            float sqrtDenom = oh + eta * ih;

            float no = max(dot(n, wo), 0.0);
            float ni = max(dot(-n, wi), 0.0);
            float nh = max(dot(n, wh), 0.0);
            float Vis_SmithVL = no * (ni * (1.0 - alpha) + alpha) +
                ni * (no * (1.0 - alpha) + alpha);
            if (Vis_SmithVL > 0.0 && nh > 0.0 && sqrtDenom > 0.0) {
                Fr = T * clamp(8.0 * oh * ni * ih * oh / nh / Vis_SmithVL /
(sqrtDenom * sqrtDenom), 0.0, 1.0);
            }
        }
}
throughput *= Fr / prob;

radiance += throughput * materials[hit_record.hit_entity_id].emission;
```

5. 原理化BSDF（Principled / Disney）

参数：固有色（albedo color，c），折射率（ior，η），金属度（metallic），粗糙度（roughness），高光（specular），高光染色（specularTint），光泽（sheen），光泽染色（sheenTint），清漆（clearcoat），清漆粗糙度（clearcoatGloss），各向异性（anisotropy），透射比（specTrans）、自发光（emission），自发光强度（emission strength）。

BSDF：参见原论文[16][17]。渲染结果参见Scene 101 with Nahida and Coffee Maker。

核心代码：参见github仓库中`/srcs/spark/shaders/disney.glsl`及其include的所有文件。

路径追踪部分使用了不同于以上材质的函数，代码如下：

```
vec3 SampleRay2(vec3 origin, vec3 direction) {
    vec3 L = vec3(0.f), beta = vec3(1.f);
    vec3 o = origin, d = direction;
    bool specularBounce = false;
    int bounces;
```

```
    // Added after book publication: etaScale tracks the accumulated effect
    // of radiance scaling due to rays passing through refractive
    // boundaries (see the derivation on p. 527 of the third edition). We
    // track this value in order to remove it from beta when we apply
    // Russian roulette; this is worthwhile, since it lets us sometimes
    // avoid terminating refracted rays that are about to be refracted back
    // out of a medium and thus have their beta value increased.
    float etaScale = 1;

    for (bounces = 0;; ++bounces) {
        // Find next path vertex and accumulate contribution

        // Intersect _ray_ with scene and store intersection in _isect_
        // SurfaceInteraction isect;
        bool foundIntersection = TraceRay(o, d);

        // Possibly add emitted light at intersection
        if (bounces == 0 || specularBounce) {
            // Add emitted light at path vertex or from the environment
            if (foundIntersection) {
                L += beta * hit_record.emission;
            } else {
                L += beta * SampleEnvmap(d);
            }
        }

        // Terminate path if ray escaped or _maxDepth_ was reached
        if (!foundIntersection || bounces >= guo.num_bounces) break;

        // Compute scattering functions and skip over medium boundaries
        BSDF bsdf;
        DisneyBSSRDF bssrdf;
        ComputeScatteringFunctions(bsdf, bssrdf, hit_record,
TransportMode_Radiance);
        if (bsdf.nBxDFs == 0) {
            o = hit_record.position;
            bounces--;
            continue;
        }

        // Sample illumination from lights to find path contribution.
        // (But skip this for perfectly specular BSDFs.)
        if (NumComponents(bsdf, BSDF_ALL & ~BSDF_SPECULAR) >
            0) {
            vec3 Ld = beta * UniformSampleOneLight(bsdf, -d);
            L += clamp(Ld, 0, guo.clamp);
        }

        // Sample BSDF to get new path direction
        vec3 wo = -d, wi;
        float pdf;
        uint flags;
        vec2 u = vec2(RandomFloat(), RandomFloat());
        vec3 f_ = Sample_f(bsdf, wo, wi, u, pdf, BSDF_ALL, flags);
        if (f_ == vec3(0) || pdf == 0.f) break;
```

```
        beta *= f_ * AbsDot(wi, hit_record.normal) / pdf;
        specularBounce = (flags & BSDF_SPECULAR) != 0;
        if ((flags & BSDF_SPECULAR) == BSDF_SPECULAR &&
            (flags & BSDF_TRANSMISSION) == BSDF_TRANSMISSION) {
            float eta = bsdf.eta;
            // Update the term that tracks radiance scaling for refraction
            // depending on whether the ray is entering or leaving the
            // medium.
            etaScale *= (dot(wo, hit_record.geometry_normal) > 0) ? (eta * eta)
: 1 / (eta * eta);
        }
        // ray = isect.SpawnRay(wi);
        o = hit_record.position;
        d = wi;

        // Possibly terminate the path with Russian roulette.
        // Factor out radiance scaling due to refraction in rrBeta.
        vec3 rrBeta = beta * etaScale;
        if (MaxComponentValue(rrBeta) < guo.rrThreshold && bounces > 3) {
            float q = max(.05, 1 - MaxComponentValue(rrBeta));
            if (RandomFloat() < q) break;
            beta /= 1 - q;
        }
    }
    return L;
}
```

（二）多重重要性采样（MIS）

直接光源采样

计算radiance时分为直接光L_d和间接光L_i，分别采样一根光线计算贡献。计算L_d时直接在光源的几何形状上均匀采样，而不是半球均匀采样。

对于非Principled材质，核心代码：

```
vec3 SampleEmission(vec3 origin, vec3 origin_normal) {
  vec3 energy = vec3(0.0);
  uint emission_id = RandomInt(guo.num_emissions);
  uint emission_entity_id = emissions[emission_id];
  ObjectInfo object_info = object_infos[emission_entity_id];
  uint face_id = RandomInt(object_info.num_faces);
  Vertex v0 = GetVertex(
      object_info.vertex_offset +
      indices[object_info.index_offset + face_id * 3 + 0]);
  Vertex v1 = GetVertex(
      object_info.vertex_offset +
      indices[object_info.index_offset + face_id * 3 + 1]);
  Vertex v2 = GetVertex(
      object_info.vertex_offset +
      indices[object_info.index_offset + face_id * 3 + 2]);
  mat3 object_to_world = mat3(entity_objects[emission_entity_id].model);
  mat3 positions = mat3(entity_objects[emission_entity_id].model *
                        mat3x4(vec4(v0.position, 1.0), vec4(v1.position, 1.0),
                               vec4(v2.position, 1.0)));
```

```
  float angle;
  vec3 wi;
  SampleSolidAngle(origin, positions, angle, wi);
  HitRecord save_hit_record = hit_record;
  if (TraceRay(origin, wi)) {
    if (hit_record.hit_entity_id == emission_entity_id &&
        ray_payload.primitive_id == face_id) {
      energy = materials[emission_entity_id].emission *
               materials[emission_entity_id].emission_strength *
               max(dot(origin_normal, wi), 0.0) * angle *
               object_info.num_faces * guo.num_emissions;
    }
  }
  hit_record = save_hit_record;
  return energy;
}
```

对于Principled材质，参见github仓库中`/srcs/spark/shaders/path_tracing.rgen`的函数`UniformSampleOneLight(...)`

BRDF采样

计算L_i时不使用半球均匀采样，而是基于$f(l,v)(n\cdot v)$使用不同的pdf采样，如漫反射表面使用$n\cdot v$采样，镜面反射表面使用$D(h)(n\cdot h)$采样。

对于非Principled材质，核心代码：

```
vec3 ImportanceSamplingLamberrian(vec3 N, vec3 T) {
  float phi = 2.0 * PI * RandomFloat();
  float theta = acos(1.0 - 2.0 * RandomFloat()) * 0.5;
  vec3 B = cross(N, T);
  vec3 W = vec3(sin(theta) * cos(phi), sin(theta) * sin(phi), cos(theta));
  vec3 wi = T * W.x + B * W.y + N * W.z;
  return wi;
}

vec3 ImportanceSamplingSpecular(vec3 N, vec3 T, float alpha) {
  float phi = 2.0 * PI * RandomFloat();
  float ksi = RandomFloat();
  float cos_theta = sqrt((1.0 - ksi) / (ksi * (alpha * alpha - 1.0) + 1.0));
  float sin_theta = sqrt(1.0 - cos_theta * cos_theta);
  vec3 B = cross(N, T);
  vec3 W = vec3(sin_theta * cos(phi), sin_theta * sin(phi), cos_theta);
  return normalize(T * W.x + B * W.y + N * W.z);
}
```

对于Principled材质，参见参见github仓库中`/srcs/spark/shaders/bsdf.glsl`的函数`Sample_f(BSDF...)`。

多重重要性采样

进行计算L_d时，同时使用光源采样和BRDF采样，记光源采样的pdf为p，L_d为L_p，BRDF采样的pdf为q，L_d为L_q，则新的L_d为

$$L_d = \frac{p^2 L_p + q^2 L_q}{p^2 + q^2}.$$

仅对Principled实现了，代码参见github仓库中`/srcs/spark/shaders/path_tracing.rgen`的函数`UniformSampleOneLight(...)`

（三）材质贴图

每一种参数都可以使用贴图替换。比如颜色贴图，参见Scene 101 with Nahida and Coffee Maker中的Nahida。

核心代码：参见github仓库中`/srcs/spark/assets/material.h`以及`/srcs/spark/shaders/material.glsl`

（三）抗锯齿

每个像素点发射光线时，不是在像素点中央发射，而是在像素点所在的正方形中均匀采样，并累加贡献。

核心代码，位于github仓库中`/srcs/spark/shaders/path_tracing.rgen/main()`：

```
void main() {
  mat4 camera_to_world = inverse(guo.camera);
  mat4 screen_to_camera = inverse(guo.projection);

  for (int i = 0; i < guo.num_samples; i++) {
    InitRandomSeed(gl_LaunchIDEXT.x, gl_LaunchIDEXT.y,
                   guo.accumulated_sample + i);
    vec2 pixelPoint = vec2(gl_LaunchIDEXT.xy) + vec2(RandomFloat(),
RandomFloat()); // 抗锯齿
    vec2 inUV = pixelPoint / vec2(gl_LaunchSizeEXT.xy);
    vec2 d = inUV * 2.0 - 1.0;

    // 景深
    vec3 c_origin = vec3(RandomInCircle() * guo.aperture, 0);
    vec3 c_target = vec3(screen_to_camera * vec4(d.x, d.y, 1, 1));
    vec3 c_focal = normalize(c_target) * guo.focal_length;
    vec3 c_direct = normalize(c_focal - c_origin);
    vec4 origin = camera_to_world * vec4(c_origin, 1);
    vec4 direction = camera_to_world * vec4(c_direct, 0);

    imageStore(accumulation_color, ivec2(gl_LaunchIDEXT.xy),
               imageLoad(accumulation_color, ivec2(gl_LaunchIDEXT.xy)) +
                   vec4(SampleRay1(origin.xyz, direction.xyz), 1.0));
    imageStore(accumulation_number, ivec2(gl_LaunchIDEXT.xy),
               imageLoad(accumulation_number, ivec2(gl_LaunchIDEXT.xy)) + 1.0);
  }
}
```

（四）景深

基于光圈（aperture，l）和焦距（focal length，f）的物理相机。对于原本的采样光线$(\mathbf{o}, \mathbf{d})$，$\mathbf{o}$为射线原点，$\mathbf{d}$为方向，计算出焦点位置$\mathbf{o} + f\mathbf{d}$，随机在光圈内采样一点$\mathbf{o}'$，新的采样光线为$(\mathbf{o}', normalize(\mathbf{o} + f\mathbf{d} - \mathbf{o}'))$。

渲染效果参见Scene 101 with Nahida and Coffee Maker。

核心代码，参见抗锯齿。

（五）OBJ文件及对应材质、贴图导入

利用助教提供的tinyobj外部库，实现将obj文件及对应材质、贴图导入scene中。由于mtl支持pbr扩展，因此正好利用。

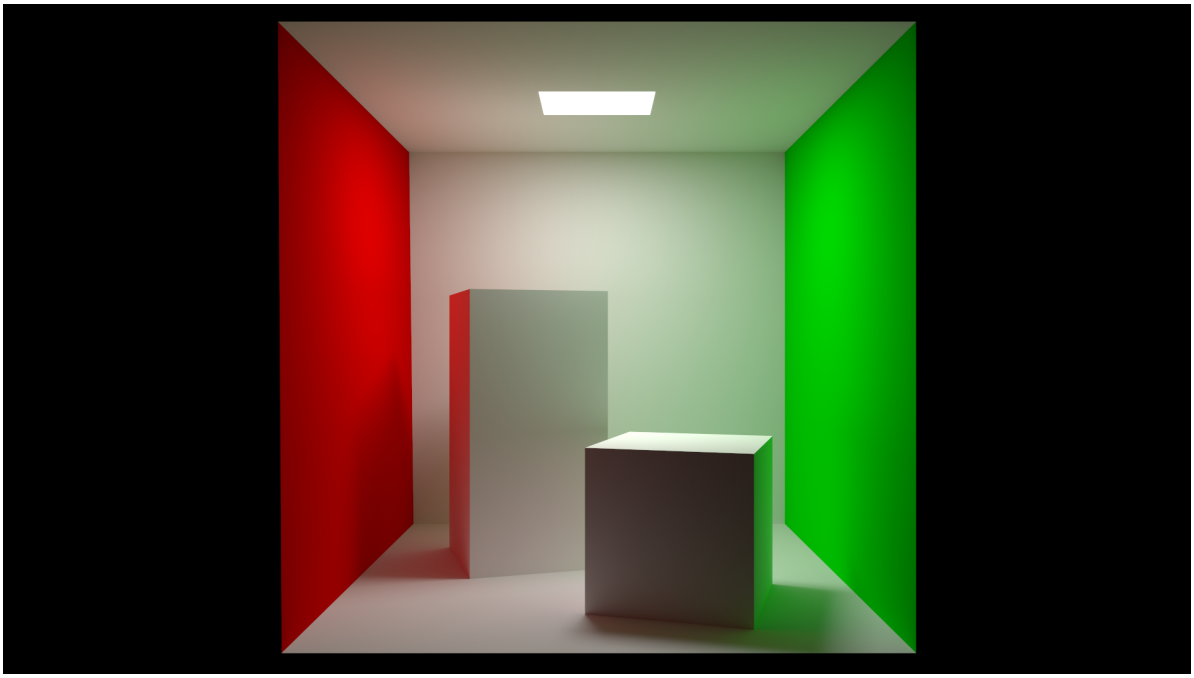
核心代码，参见github仓库中`/srcs/spark/assets/scene.cpp/Scene::LoadObjFile(const std::string &file_path, const glm::mat4 &transform = glm::mat4{1.0f})`

同时可以利用xml文件指定文件位置，代码：

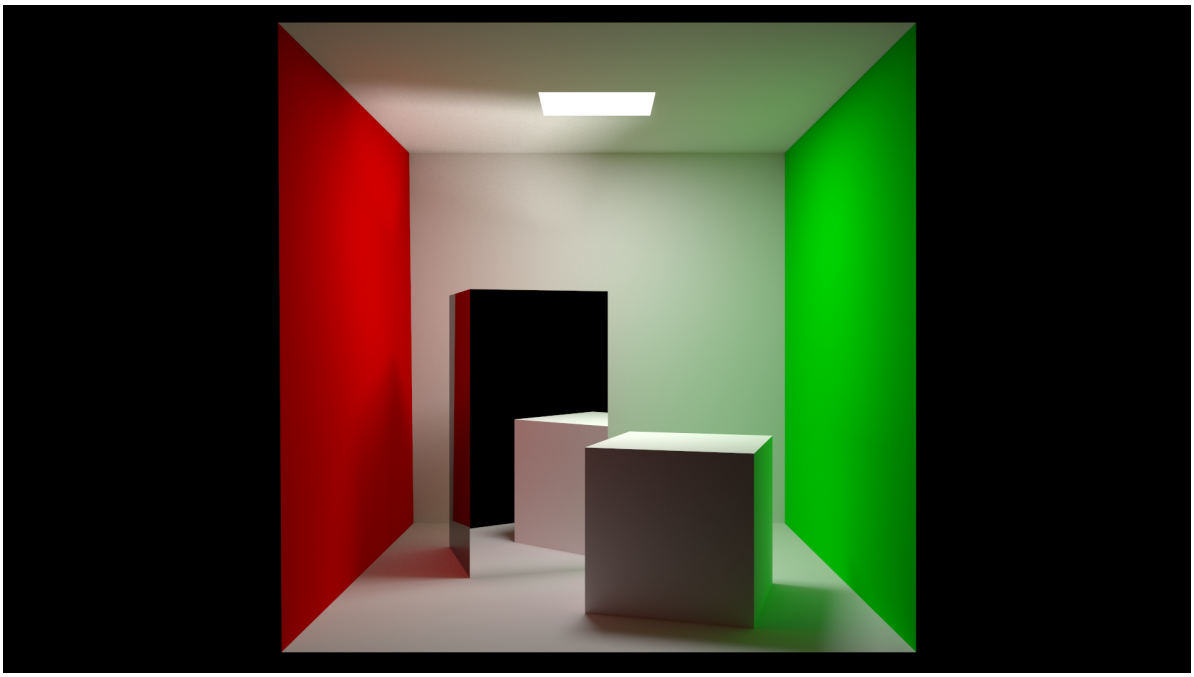
```
Scene::Scene(const std::string &filename) : Scene() {
    if (...) {
        ...
    } else if (element_type == "scene") {
        LoadObjFile(child_element);
    } else {
        ...
    }
}
```

三、渲染结果

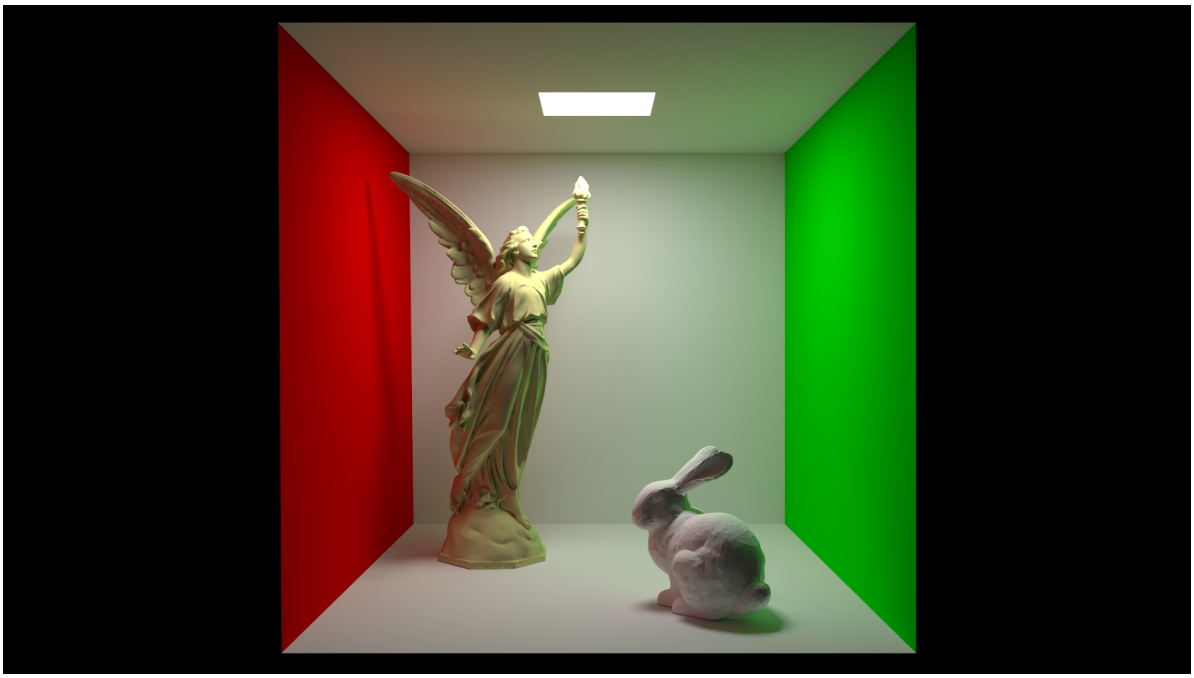
Cornell Box



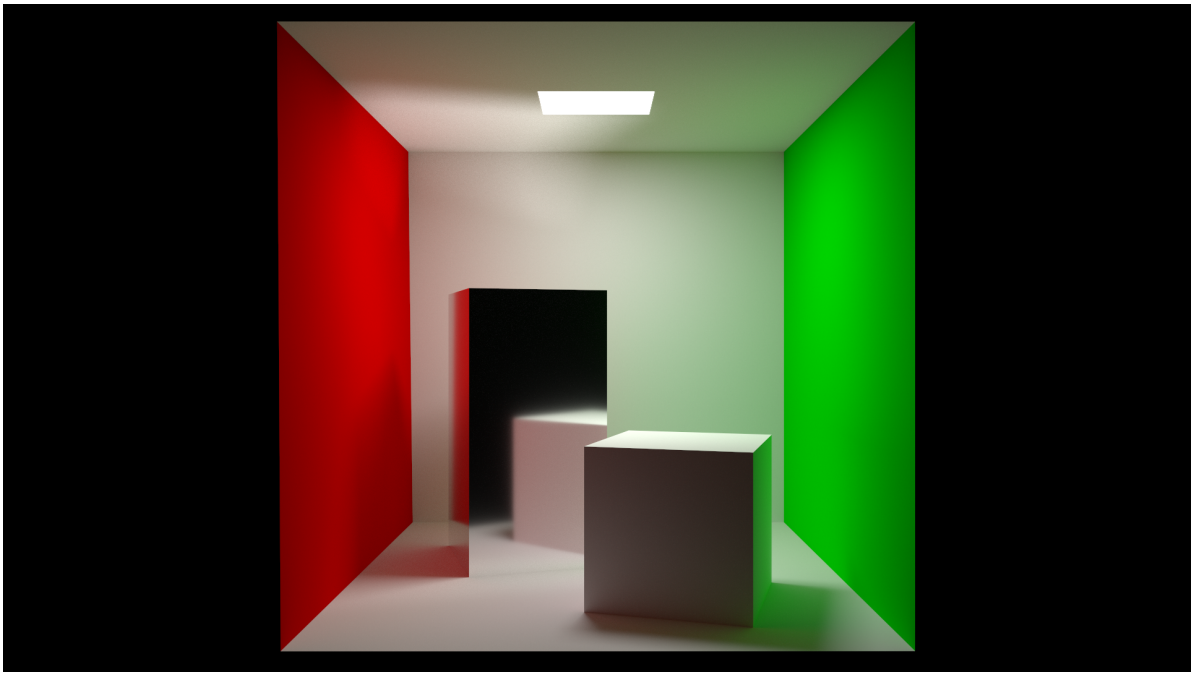
Cornell Box Specular



Cornell Box with Lucy and Bunny

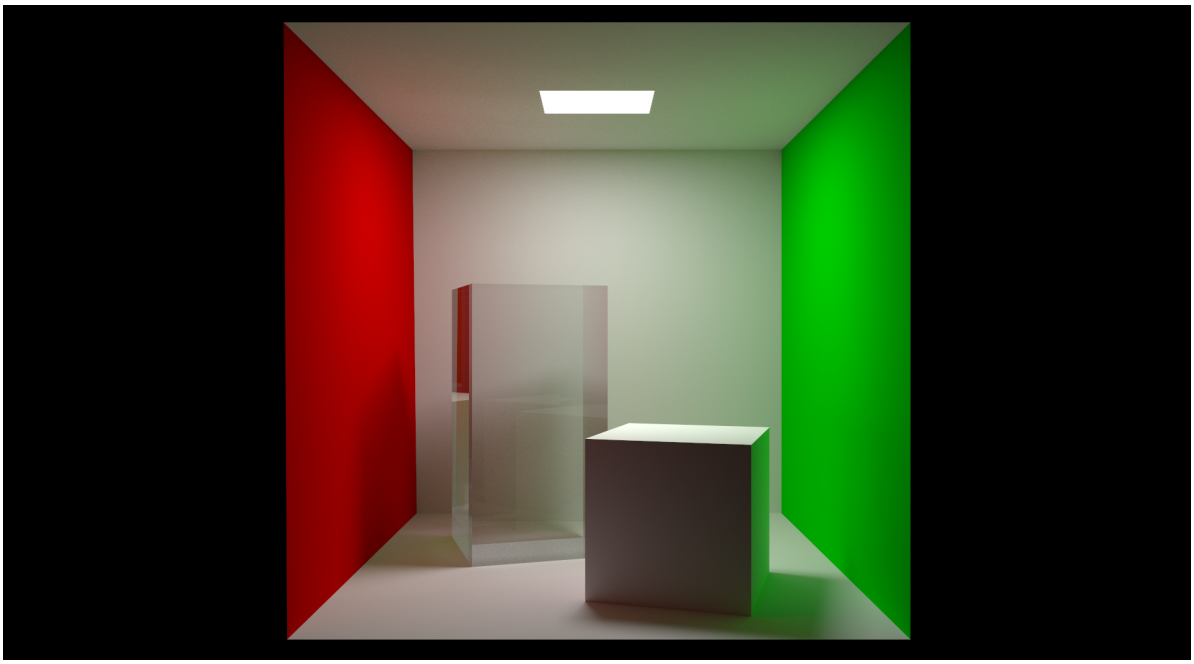


Cornell Box Rough Specular



$roughness = 0.1,\ F_0 = (1, 1, 1)$

Cornell Box Transmissive



$ior = 1.5, roughness = 0, T = (1, 1, 1)$

Cornell Box with Specular Lucy and Transmissive Bunny

Lucy: $roughness = 0.2, F_0 = (0.615, 0.574, 0.249)$

Bunny: $ior = 1.5, roughness = 0.2, T = (1, 1, 1)$

Scene 101 with Nahida and Coffee Maker

roughness默认为0.5，ior默认为1.45，specular默认为0.5，clearcoatGloss默认为0.03，其他参数默认为0

Scene 101

- 场景来源：[8]
- 红色透明台阶：roughness=0.1，ior=1.5，specTrans=1
- 大球：metallic=0.7
- 中球：metallic=0.7，anisotropy=0.7
- 小球：metallic=0.7，roughness=0.2
- 青色清漆弧形墙壁：clearcoat=1，clearcoatGloss=0.03

- 下层粉色地面：sheen=1
- 上层粉色地面：roughness=0
- 紫色背景墙：metallic=0.318182
- 青色杯子：roughness=0.213636

Nahida

- 模型来源：[10]
- 均为默认参数，使用了模型自带的颜色贴图

Coffee Maker

- 模型来源：[9]
- 橙色：roughness=0，specular=0.8
- 黑色：roughness=0.316228，specular=0.475758
- 玻璃：specTrans=1
- 金属：metallic=1，roughness=0.158114

四、代码仓库

链接：https://github.com/RY5228/sparkium-pub

五、待优化的部分

Principled部分基于pbrt v3[6]的代码翻译完成。由于其使用了OOP（基类、继承等），此方法不适用于GPU，因此进行代码翻译时使用了很多判断，增加了很多计算量。pbrt v3将环境贴图视为infinity light，而助教提供的框架不是这样做的，因此未实现环境光。（其余材质完成了环境光）

Principled没有翻译次表面采样的部分。

Principled和其他的材质使用了不同的SampleRay函数，没有合为统一的函数。

六、参考文献

[1] ybbbt，蒙特卡洛路径追踪

[2] 孙小磊，基于物理的渲染：微平面理论(Cook-Torrance BRDF推导)

[3] 孙小磊，重要性采样和多重重要性采样在路径追踪中的应用

[4] 222先生，BRDF重要性采样

[5] 浅墨（毛星云），基于物理的渲染（PBR）白皮书 | PBR White Paper

[6] Matt Pharr，pbrt, Version 3

[7] Blender，原理化BSDF

[8] sabazavr，Scene 101

[9] cekuhnen，Studio Rendering: Braun KS 20 Coffee Maker

[10] 原神，【原神】纳西妲

[11] Cornell University，Cornell Box Data

[12] AirGuanZ，Disney Principled BRDF实现笔记

[13] 小小木同子，【渲染】Disney BSDF 深度解析

[14] AirGuanZ，基于光线追踪的渲染中景深（Depth of field）效果的实现

[15] Exocortex，Extending Wavefront MTL for Physically-Based Rendering

[16] Burley B, Studios W D A. Physically-based shading at Disney[C]//ACM SIGGRAPH. 2012, 2012: 1-7.

[17] Burley, B. (2015). Extending the Disney BRDF to a BSDF with integrated subsurface scattering. *SIGGRAPH Course: Physically Based Shading in Theory and Practice. ACM, New York, NY*, *19*.