

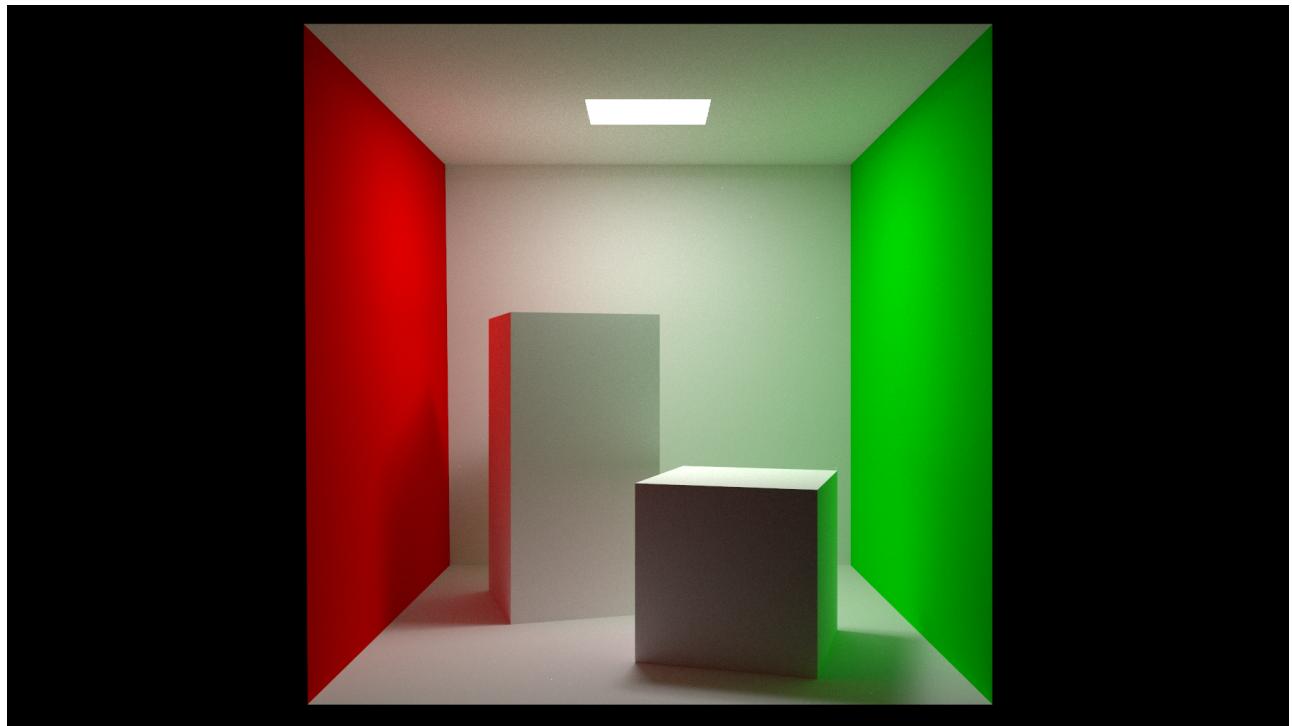
Advanced Computer Graphics: Final Report

Yanpeng Wei, 2021010795

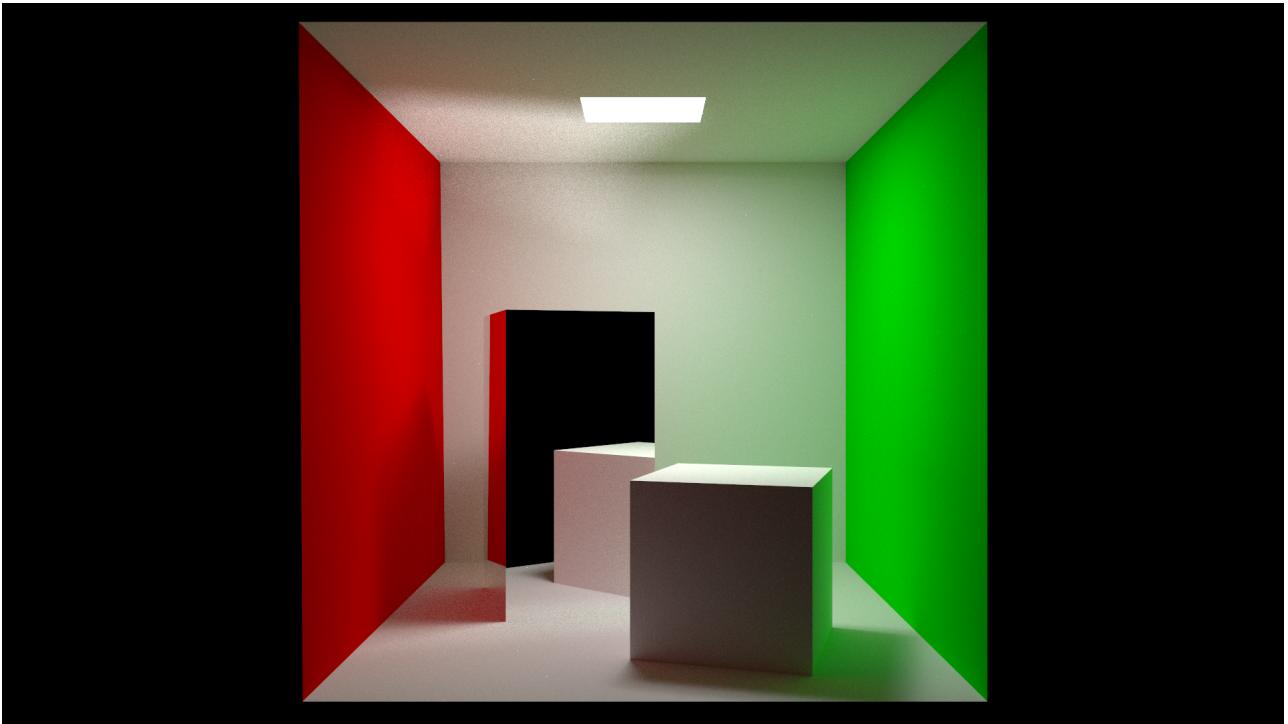
Yiyin Wang, 2020011604

Base

Cornell Box



Specular



Lucy & Bunny



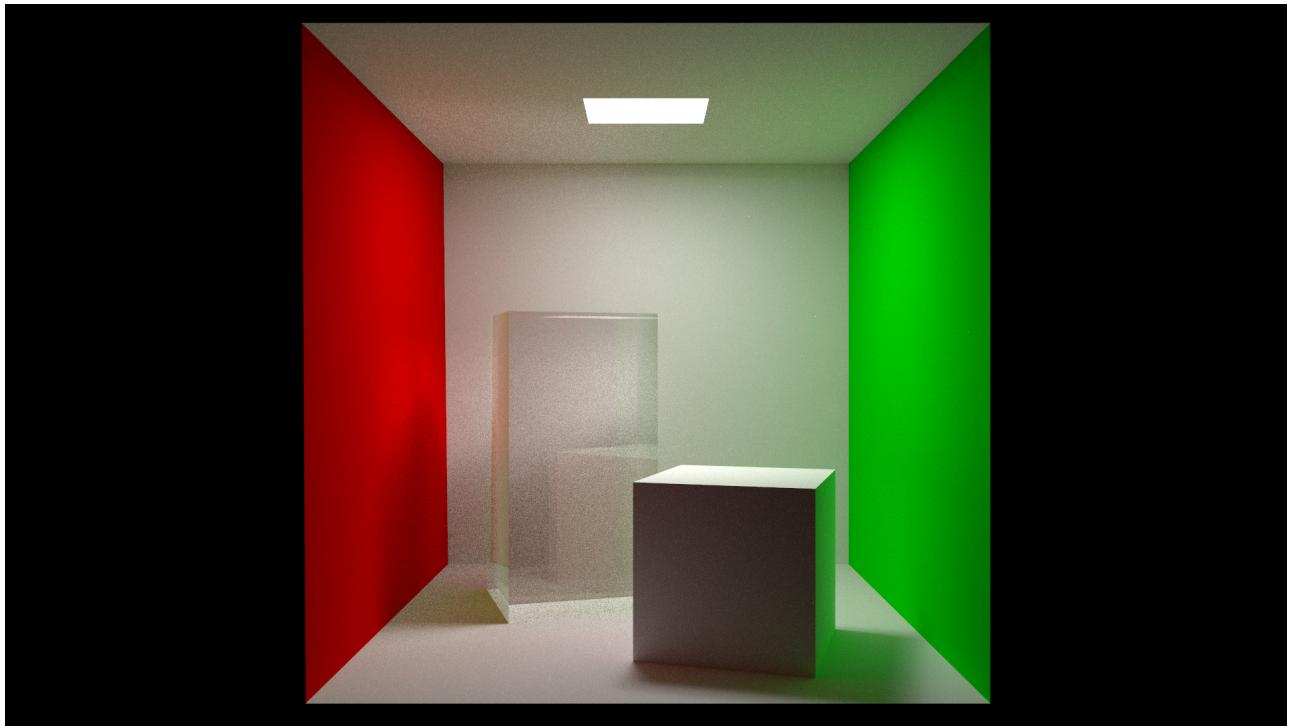
Transmissive

折射材质，默认真空折射率为 1.0, 给定 iot (Transmissive Rate), 根据 Fresnel Form 决定是否折射。

Fresnel Form 采用近似:

```
1 float f0 = pow((eta_i - eta_o) / (eta_i + eta_o), 2);  
2 float fresnel = f0 + (1.0f - f0) * pow(1.0 - abs(cos_theta), 5);
```

iot = 2.0:



核心代码:

```
1 // hit_record.normal = hit_record_normal;  
2 vec3 median = -dot(direction, hit_record_normal) *  
hit_record_normal;  
3 vec3 proj_x = normalize(direction + median);  
4 vec3 proj_y = hit_record_normal;  
5 // theta: 初始角度, phi: 穿过后角度  
6 float cos_theta = dot(hit_record_normal, direction);  
7 float sin_theta = sqrt(1-cos_theta*cos_theta);  
8 float sin_phi, cos_phi;  
9 float eta_i = 1.0;  
10 float eta_o = material.iot;  
11 float eta = 1.0 / material.iot;  
12 if (cos_theta < 0) {  
13     proj_y = -proj_y;  
14     cos_theta = -cos_theta;  
15 } else{  
16     eta = 1.0 / eta;  
17     eta_i = material.iot;  
18     eta_o = 1.0;
```

```

19 }
20 float f0 = pow((eta_i - eta_o) / (eta_i + eta_o), 2);
21 float fresnel = f0 + (1.0f - f0) * pow(1.0 - abs(cos_theta), 5);
22 if(RandomFloat() > fresnel) {
23     sin_phi = eta * sin_theta;
24     cos_phi = sqrt(1-sin_phi*sin_phi);
25     origin = hit_record.position;
26     direction = cos_phi * proj_y + sin_phi * proj_x;
27 } else {
28     vec3 wi = direction - 2.0f * dot(direction, hit_record.normal)
29             * hit_record.normal;
30     // this is calculate shading
31     origin = hit_record.position;
32     direction = wi;
33 }
```

微表面模型： cook-torrance brdf

具体公式见 [链接](#)，加入了 α 参数作为粗糙度， $k = \frac{\alpha^2}{2}$ 。效果见自定义场景。

核心代码，F0 直接利用了 albedo:

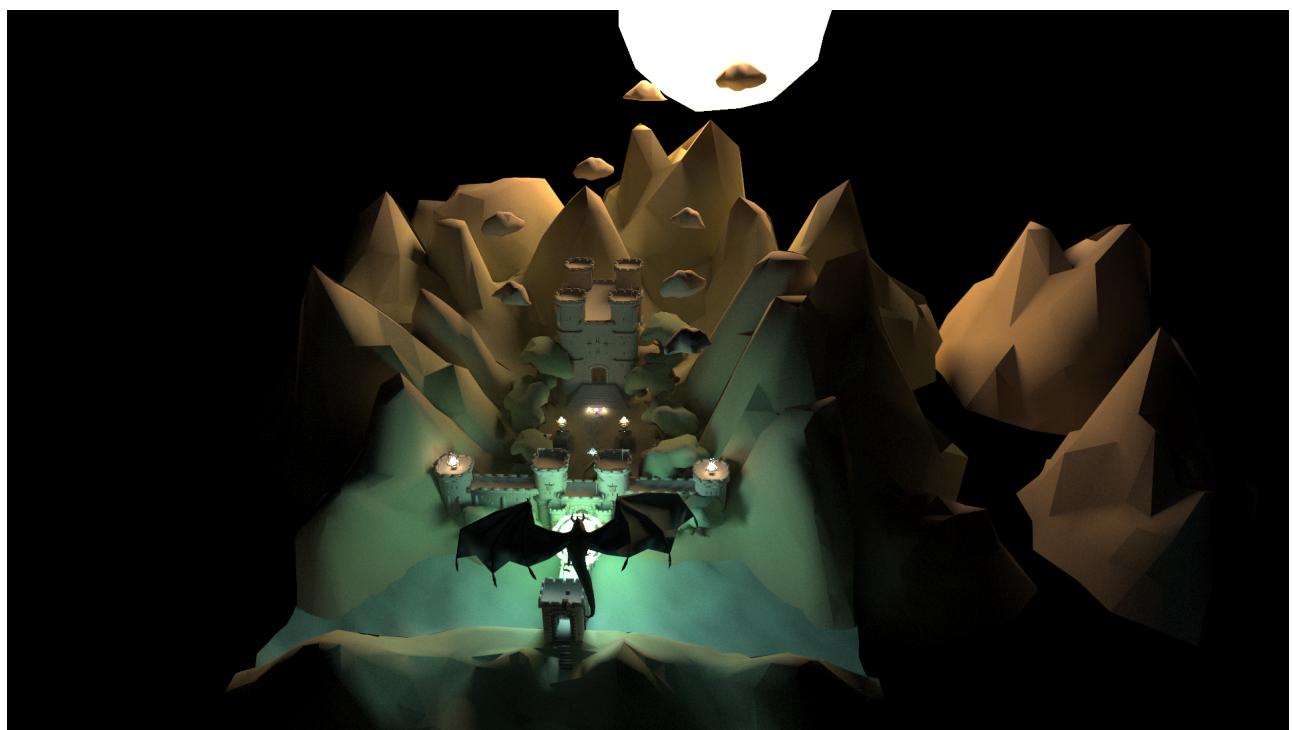
```

1 vec3 calculate_brdf(vec3 n, vec3 wi, vec3 wo, float D, vec3 h,
float alpha, vec3 F0) {
2     // float roughness = alpha;
3     // float k = pow(roughness + 1.0f, 2) / 8.0f;
4     float k = (alpha * alpha) / 2.0f;
5     float G_i = dot(n, wi) / (dot(n, wi) * (1.0f - k) + k);
6     float G_o = dot(n, wo) / (dot(n, wo) * (1.0f - k) + k);
7     vec3 fresnel = F0 + (vec3(1.0f) - F0) * pow(1.0f - dot(h, wi),
5.0f);
8     float G = G_i * G_o;
9     // if(alpha == 0.0f) {
10     //     return fresnel * D * dot(wi, n);
11     // }
12     // if(dot(n, wi) < 0.0f || dot(n, wo) < 0.0f) return 0.1f;
13     return (fresnel * G * D / (4.0f * dot(n, wi) * dot(n, wo))) *
14         dot(wi, n);
```

Texture

实现了发光贴图和颜色贴图。

同时，我们将一个同时有两种贴图的同一物体做在了一个场景，用于展示：



(整个场景只有一个 object，见 `new_copy.xml`)

核心代码：

```
1 if (material.material_type != MATERIAL_TYPE_EMISSION) {  
2     vec3 emission = material.emission *  
3  
4     vec3 texture(texture Samplers[material.emission_texture_id], hit_r  
ecord.tex_coord);  
5     if(emission != vec3(0.0f)) {  
6         radiance += throughput * emission *  
7             material.emission_strength;  
8         break;  
9     }  
10 }
```

Multiple Importance Sampling

我们对于粗糙平面，我们利用 cosine-weighted sampling，对于微表面模型，我们使用 GGX sampling。

directly sampling the light:

```
1 int n_lights = int(light_sources[0]);
2 if(n_lights != 0 && (material.material_type ==
MATERIAL_TYPE_LAMBERTIAN || material.material_type ==
MATERIAL_TYPE_PRINCIPLED)) {
3     int r = RandomInt(n_lights) + 1;
4     int face_cnt = light_sources[2 * r + 1];
5     int obj = light_sources[2 * r];
6     int faceid = RandomInt(face_cnt);
7     // vec3 v1, v2, v3;
8     ObjectInfo obj_info = object_infos[obj];
9     vertex nd0 = GetVertex(obj_info.vertex_offset +
indices[obj_info.index_offset + faceid * 3 + 0]),
10        nd1 = GetVertex(obj_info.vertex_offset +
indices[obj_info.index_offset + faceid * 3 + 1]),
11        nd2 = GetVertex(obj_info.vertex_offset +
indices[obj_info.index_offset + faceid * 3 + 2]);
12     vec3 d = vec3(RandomFloat(), RandomFloat(), RandomFloat());
13     while(d.x + d.y + d.z == 0.0f) {
14         vec3 d = vec3(RandomFloat(), RandomFloat(), RandomFloat());
15     }
16     d /= d.x + d.y + d.z;
17     d /= d.x + d.y + d.z;
18     vec3 pt = d.x * nd0.position + d.y * nd1.position + d.z *
nd2.position;
19     vec3 N = cross(nd1.position - nd0.position, nd2.position -
nd0.position);
20     float S = length(N) / 2.0f;
21     N = normalize(N);
22     vec3 origin = hit_record.position;
23     vec3 wi = normalize(pt - hit_record.position);
24     if(dot(wi, N) > 0.0f) {
25         N = -N;
26     }
27     TraceRay_0(origin, wi);
28     if(ray_payload.t < 0.0f) {
29 }
```

```

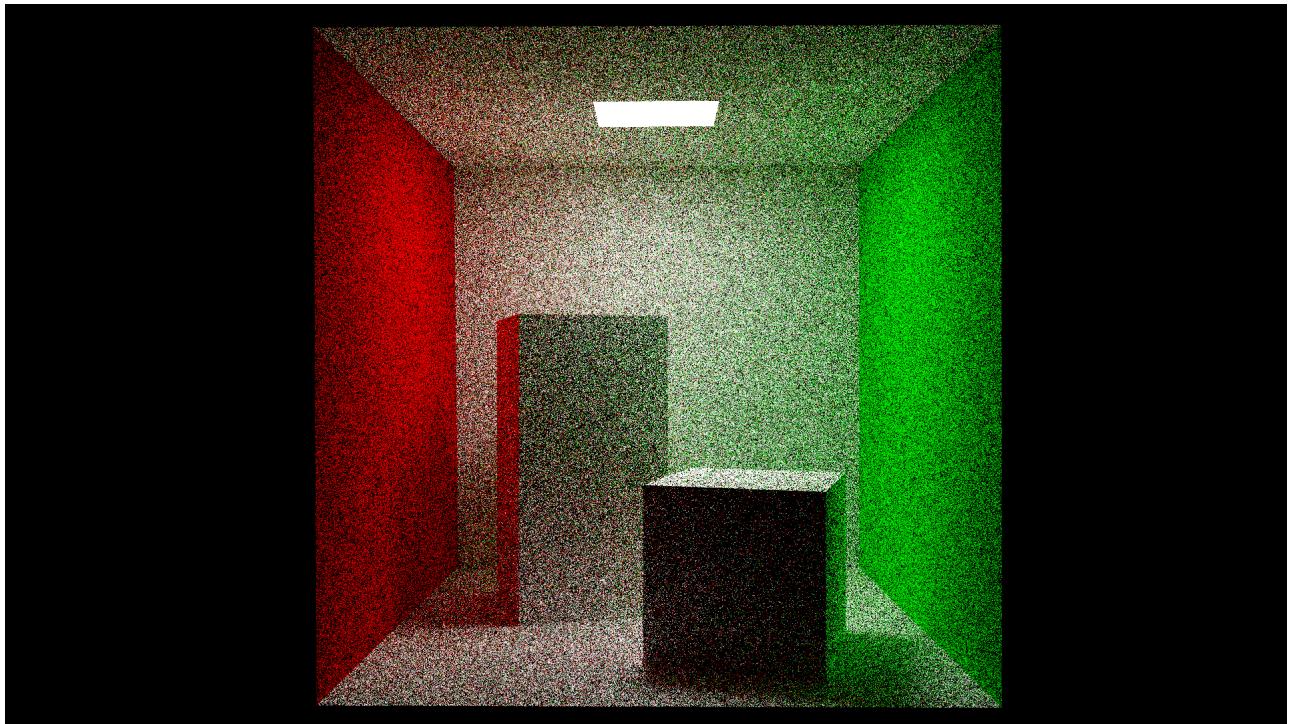
30 } else {
31     Material light_m = materials[hit_record_0.hit_entity_id];
32     if(hit_record_0.hit_entity_id == obj &&
33         light_m.material_type == MATERIAL_TYPE_EMISSION) {
34         vec3 radiance_dir = light_m.emission *
35             light_m.emission_strength
36             * max(0.0f, dot(hit_record.normal, wi)) // cosine
37             * dot(N, -wi) // cosine
38             * S / pow(length(hit_record_0.position -
39             origin), 2)
40             * n_lights * face_cnt;
41     vec3 L_direct = throughput * (material.albedo_color / PI) *
42
43         vec3(texture(texture Samplers[material.albedo_texture_id], hit_re-
44         cord.tex_coord)) * radiance_dir;
45     if(material.material_type == MATERIAL_TYPE_PRINCIPLED) {
46         if(dot(hit_record.normal, wi) > 0) {
47             float DD = D_alpha(wi, hit_record.normal, normalize(wi -
48             direction), material.alpha);
49             vec3 wo = -direction;
50             vec3 h = normalize(wi + wo);
51             if(material.alpha == 0.0f) h = hit_record.normal;
52             L_direct = throughput *
53                 texture(texture Samplers[material.albedo_texture_id], hit_re-
54                 cord.tex_coord)) * radiance_dir *
55                 calculate_brdf(hit_record.normal, wi, wo, DD, h,
56                 material.alpha, material.albedo_color) / dot(hit_record.normal,
57                 wi);
58         } else
59             L_direct = vec3(0.0f);
60     }
61     float sampling_the_light_pdf = dot(N, -wi) // cosine
62             * S / pow(length(hit_record_0.position -
63             origin), 2)
64             * n_lights * face_cnt;
65     if(sampling_the_light_pdf != 0.0f) {
66         sampling_the_light_pdf = 1.0f / sampling_the_light_pdf;
67     }
68     // if(sampling_the_light_pdf != 0.0f)
69     if(material.material_type == MATERIAL_TYPE_LAMBERTIAN)

```

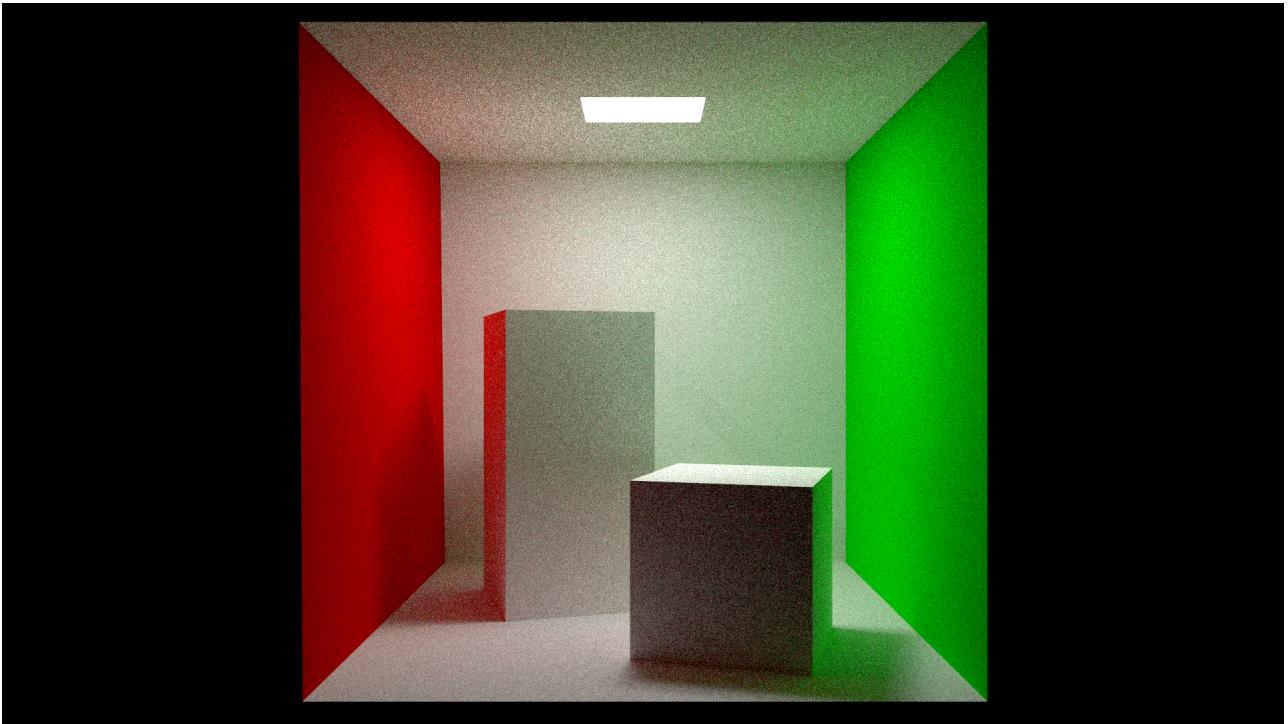
```
58         radiance += L_direct * get_pdf_ratio(Origin, wi,
59             hit_record.normal, -direction, 0);
60         else
61             radiance += L_direct * get_pdf_ratio_principled(Origin,
62                 wi, hit_record.normal, -direction, material.alpha, 0);
63     }
```

对于 MIS，我们会统计 directly sampling 和当前sample 方式的 pdf ratio w_1, w_2 ，最终会按 $w_i^2/(w_1^2 + w_2^2)$ 分配光源的贡献。

50ssp, without MIS:



50ssp, with MIS:



可以看到，收敛更快（我们同时可以从Specular里看出，因为镜面反光没有 Importance Sampling）

景深 & 抗锯齿

景深可以修改 aperture。

效果见自定义场景。

核心代码：

```
1 for (int i = 0; i < guo.num_samples; i++) {
2     InitRandomSeed(uint(pixelCenter.x), uint(pixelCenter.y),
3                     guo.accumulated_sample + i);
4
5     // vec4 origin = camera_to_world * vec4(0, 0, 0, 1);
6     // vec4 target = screen_to_camera * vec4(d.x, d.y, 1, 1);
7     // vec4 direction = camera_to_world *
8     vec4(normalize(target.xyz), 0);
9     vec2 pixelPoint = vec2(gl_LaunchIDEXT.x + RandomFloat(),
10                           gl_LaunchIDEXT.y + RandomFloat());
11    vec2 inUV = pixelPoint / vec2(gl_LaunchSizeEXT.x,
12                                   gl_LaunchSizeEXT.y);
13    vec2 d = inUV * 2.0 - 1.0;
14    vec3 o_ = vec3(RandomInCircle() * guo.aperture, 0); // origin
```

```

12     vec3 t_ = vec3(screen_to_camera * vec4(d.x, d.y, 1, 1)); // target
13     vec3 f_ = normalize(t_) * guo.focal_length; // focal
14     vec3 d_ = normalize(f_ - o_); // direct
15     vec4 origin = camera_to_world * vec4(o_, 1);
16     vec4 direction = camera_to_world * vec4(d_, 0);
17     imageStore(accumulation_color, ivec2(gl_LaunchIDEXT.xy),
18                 imageLoad(accumulation_color,
19                             ivec2(gl_LaunchIDEXT.xy)) +
20                             vec4(sampleRay(origin.xyz, direction.xyz),
21                                  1.0));
21     imageStore(accumulation_number, ivec2(gl_LaunchIDEXT.xy),
22                 imageLoad(accumulation_number,
23                             ivec2(gl_LaunchIDEXT.xy)) + 1.0);
22 }

```

自定义场景

```

1 arch: F0 = "1.00 0.71 0.29", alpha = 0.1
2 knight: F0 = "0.05 0.05 0.05", alpha = 0.05
3 Bunny: F0 = "0.56 0.57 0.58", alpha = 0.2
4 ground: alpha = 0.01

```



加入景深:

```
1 <aperture value="0.2"/>
2 <focal_length value="44"/>
```

