Screen Space Ray Tracing

Mengzhu Wang

2024 年 5 月 19 日

1 Overview

全局光照等于直接光照加间接光照，在实时渲染中全局光照考虑直接光照和比直接光照多一次 bounce 的间接光照。为完成屏幕空间下的全局光照效果，基于需要的信息不同我们会分三阶段着色。第一次着色负责计算 Shadow Map 所需的深度值并保存到贴图中。第二次着色负责计算屏幕空间中，每个像素对应的世界坐标系下位置、世界坐标系下法线、漫反射反射率和可见性信息并最终保存到对应贴图中。第三次着色基于之前得到的场景几何信息 (像素对应的位置，法线)，场景与光源的遮挡信息 (光源坐标系的深度值)，场景的材质信息 (漫反射反射率)，来计算两次反射的全局光照结果。

一般情况下，在第二次着色时计算直接光照信息并保存下来，在之后计算间接光照时进行查询，这里转移到第三次着色中完成。

2 Direct Lighting

直接光照采用理想漫反射模型 Lambert 光照模型，其公式为

$$I_d = k_d I \cos\theta$$

`EvalDiffuse(wi,wo,uv)` 函数返回 BSDF 值即 $k_d \cos\theta$。对于漫反射来说，$f_{lambert}$ 是常数，$L_i(p,\omega_i)$ 在每个点的结果也是一样的，所以将这两项提出去，得到渲染方程

$$L_o(p,\omega_o) = L_i(p,\omega_i) f_{lambert} \int_{\Omega^+} \cos\theta_i \mathrm{d}\omega_i$$

又 $\mathrm{d}\omega = \sin\theta \mathrm{d}\theta\mathrm{d}\phi$，可得 $\int_{\Omega^+} \cos\theta \mathrm{d}\omega = \int_{\theta=0}^{\frac{\pi}{2}} \int_{\phi=0}^{2\pi} \cos\theta \sin\theta \mathrm{d}\theta\mathrm{d}\phi = \pi$。且漫反射率 albedo 为反射光和入射光之比，因此

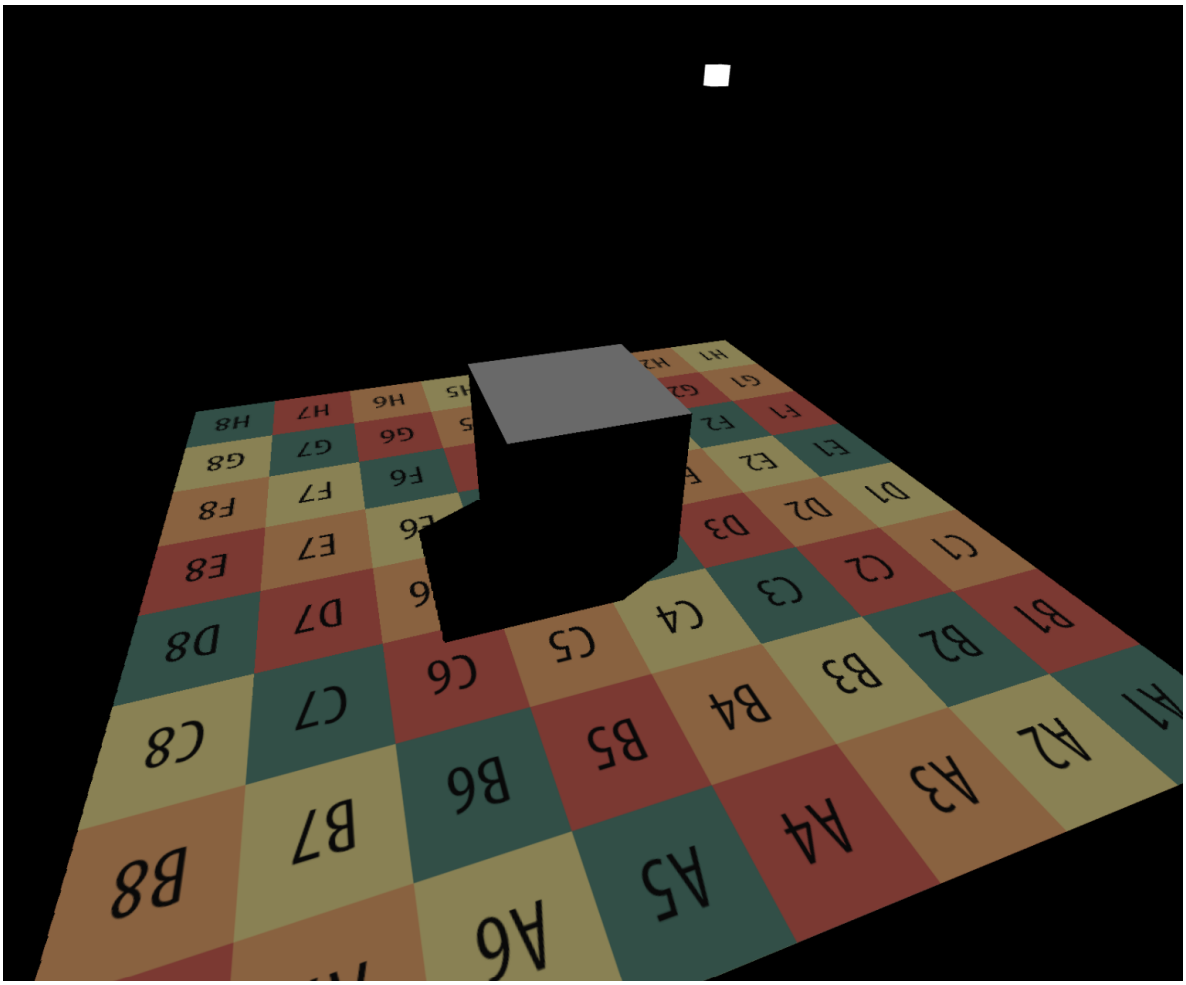
$$f_{lambert} = \frac{albedo}{\pi}$$

```
vec3 EvalDiffuse(vec3 wi, vec3 wo, vec2 uv) {
  vec3 albedo = GetGBufferDiffuse(uv);
```

```
  vec3 normal = GetGBufferNormalWorld(uv);
  float cos = max(0.0, dot(normal, wi));
  return albedo * cos * INV_PI;
}
```

EvalDirectionalLight 负责计算着色点位于 uv 处得到的光源的辐射度，并考虑遮挡关系。

```
vec3 EvalDirectionalLight(vec2 uv) {
  vec3 Le = GetGBufferuShadow(uv) * uLightRadiance;
  return Le;
}
```



3 Screen Space Ray Tracing

SSR（Screen Space Reflection）是在屏幕空间做光线追踪，光线与相机看到的一层场景求交，找到交点后计算对着色点的贡献。求反射光线与场景的相交是其中最为关键的问题之一。最基础的

方法是线性光线步进（Linear Raymarch）。该方法沿着反射方向以固定的步长前进，每次都将当前深度和场景的深度比较。若当前深度更浅，则继续前进，否则停止。

RayMarch 函数返回值为是否相交，若相交则将 hitPos 设置为交点。从参数 ori 开始，向着 dir 方向前进一定距离，直至相交。

```
bool RayMarch(vec3 ori, vec3 dir, out vec3 hitPos) {
  vec3 step = dir * 0.01;
  vec3 cur = ori;
  for (int i = 0; i < 150; i++) {
    vec2 uv = GetScreenCoordinate(cur);
    float gbufferDepth = GetGBufferDepth(uv);
    float rayDepth = GetDepth(cur);

    if (rayDepth > gbufferDepth + 0.0001) {
      hitPos = cur;
      return true;
    }
    cur += step;
  }
  return false;
}
```

可以用镜面反射测试步进算法是否正确。

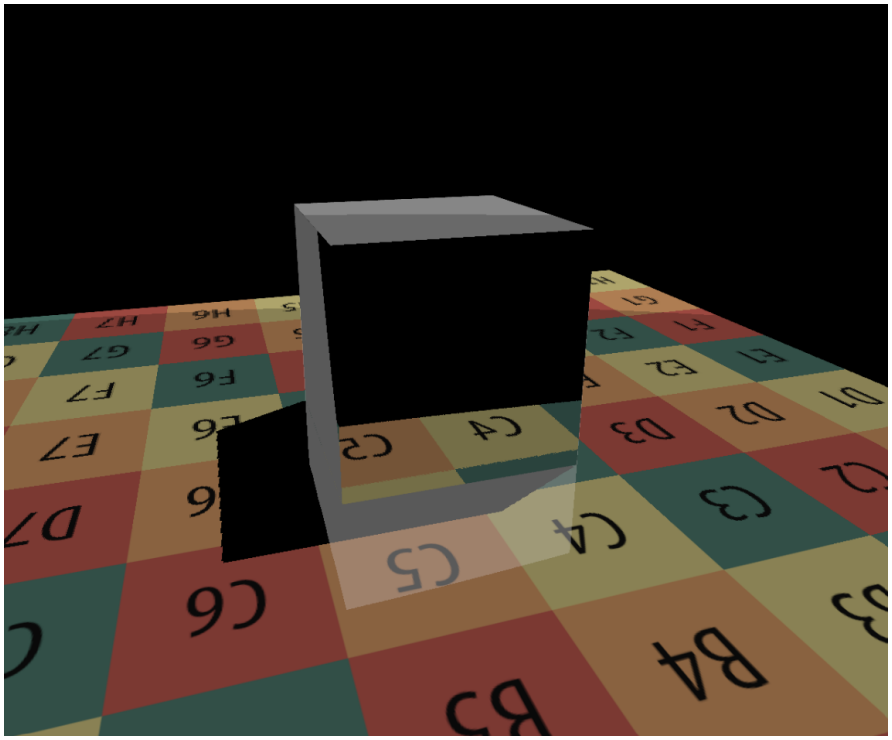
```
vec3 EvalSSR(vec3 wi, vec3 wo, vec2 uv) {
  vec3 normal = GetGBufferNormalWorld(uv);
  vec3 reflectdir = normalize(reflect(-wo, normal));
  vec3 hitPos;
  if (RayMarch(vPosWorld.xyz, reflectdir, hitPos)) {
    vec2 screenUV = GetScreenCoordinate(hitPos);
    return GetGBufferDiffuse(screenUV);
  } else {
    return vec3(0.);
  }
}

void main() {
  float s = InitRand(gl_FragCoord.xy);
  vec3 L = vec3(0.0);

  vec3 wi = normalize(uLightDir);
  vec3 wo = normalize(uCameraPos - vPosWorld.xyz);
```

```
  vec2 uv = GetScreenCoordinate(vPosWorld.xyz);
  vec3 normal = GetGBufferNormalWorld(uv);
  L = EvalDiffuse(wi, wo, uv) * EvalDirectionalLight(uv);
  L += EvalSSR(wi, wo, uv) * 0.2;

  vec3 color = pow(clamp(L, vec3(0.0), vec3(1.0)), vec3(1.0 / 2.2));
  gl_FragColor = vec4(vec3(color.rgb), 1.0);
}
```



4 Indirect Lighting

计算间接光照采用蒙特卡洛方法求解渲染方程。采样光线方向时，可以采用均匀采样上半球的函数 `SampleHemisphereUniform(inout s, out pdf)` 或者按 cos 值加权采样上半球，其对应的函数为 `SampleHemisphereCos(inout s, out pdf)`，返回一个局部坐标系的位置。接着 `LocalBasis` 函数通过传入的世界坐标系法线建立局部坐标系，返回两个切线向量。通过它们可以将采样得到的局部坐标系的方向转换到世界坐标系中。若采样光线与场景有交点，就需要计算间接光照，实际这里从某个着色点出发寻找有无提供间接光照的着色点。position0 的 EvalDiffuse 的光线入射方向是间接光源到该着色点的方向，而出射方向则仍然是 position0 到相机的方向。position1 的入射方向是真正光源的方向，出射方向是步进方向。最后需要把累加的间接光除以采样数取得平均值。

Algorithm 1 间接光照

$L_{indirect} \leftarrow 0$
for $i = 0$ to SAMPLE_NUM **do**
  $direction \leftarrow SampleDirection()$
  $hit, position_1 \leftarrow Intersection()$
  if hit **then**
    $L \leftarrow \frac{EvaluateBSDF(position_0)}{Pdf_{BSDF}} * EvaluateBSDF(position_1) * EvaluateLight(position_1)$
    $L_{indirect} \leftarrow L_{indirect} + L$
  end if
end for
$L_{indirect} \leftarrow \frac{L_{indirect}}{\text{SAMPLE\_NUM}}$

```
void main() {
  float s = InitRand(gl_FragCoord.xy);

  vec3 L = vec3(0.0);
  vec3 L_ind = vec3(0.0);

  vec3 wi = normalize(uLightDir);
  vec3 wo = normalize(uCameraPos - vPosWorld.xyz);
  vec2 uv = GetScreenCoordinate(vPosWorld.xyz);
  vec3 normal = GetGBufferNormalWorld(uv);
  L = EvalDiffuse(wi, wo, uv) * EvalDirectionalLight(uv);

  for (int i = 0; i < SAMPLE_NUM; i++) {
    float pdf;
    vec3 b1, b2;
    vec3 local_dir = SampleHemisphereCos(s, pdf);
    LocalBasis(normal, b1, b2);
    vec3 dir = normalize(mat3(b1, b2, normal) * local_dir);

    vec3 hitPos;
    if (RayMarch(vPosWorld.xyz, dir, hitPos)) {
      vec2 hit_uv = GetScreenCoordinate(hitPos);
      L_ind += EvalDiffuse(dir, wo, uv) / pdf * EvalDiffuse(wi, dir, hit_uv) * EvalDirectionalLi
    }
  }
  L_ind /= float(SAMPLE_NUM);
  L = L + L_ind;

  vec3 color = pow(clamp(L, vec3(0.0), vec3(1.0)), vec3(1.0 / 2.2));
```

```
  gl_FragColor = vec4(vec3(color.rgb), 1.0);
}
```