全局光照技术

1 概要

全局光照（Global Illumination，GI），包含直接光照和间接光照。直接光照指光线从光源出发，直接打在物体上的效果；间接光照指光线经过其他物体反射打在当前物体上的效果。

2 Reflective Shadow Mapping（RSM）

2.1 核心思想

RSM 是为了解决间接光照问题，其核心是在被光源照亮的区域构建虚拟点光源（Virtual Point Light, VPL），然后将虚拟点光源对着色点的贡献作为间接光照的结果。

2.2 算法步骤

（1）从光源视角对场景进行渲染（类似 shadow map），存储每个像素的深度 d_p、世界空间位置 x_p、法线 n_p 和反射的光辐射通量 Φ_p，此处每个像素对应一个 VPL。

（2）用 VPL 计算间接光照，如图1所示，虚拟点光源 p 对 x（x 的法线为 n）的间接光照贡献计算公式如下：

$$E_p(x,n) = \Phi_p \frac{max(0, dot(n_p, x - x_p) max(0, dot(n, x_p - x)))}{\|x - x_p\|^4}$$

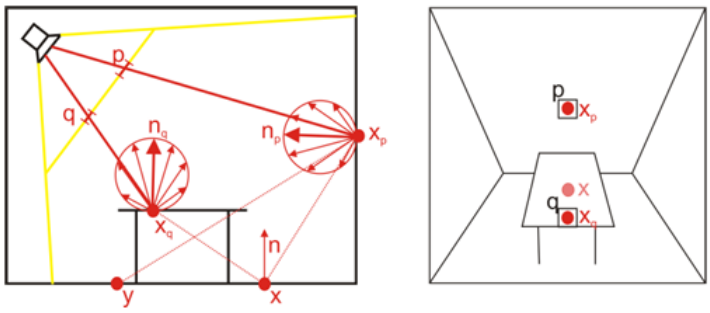

Figure 2: Two indirect pixel lights x_p and x_q corresponding to two RSM pixels p and q

图 1: RSM

2.3 优化策略

2.3.1 减少 VPL 计算

如果 RSM 的分辨率为 512×512，那么就会有 512×512 个 VPL，计算量庞大。

作者基于一个假设：越靠近着色点的 VPL，对着色点的 GI 贡献越大

对于着色点 x，它在 RSM 的坐标为 (s,t)，那么采样 VPL 的策略为（如图2）：

$$(s + r_{max}\xi_1 sin(2\pi\xi_2), t + r_{max}\xi_1 cos(2\pi\xi_2))$$

注：需要对变动的采样密度进行补偿，对每个 sample 乘以 ξ_1^2。

2.3.2 屏幕空间插值

在相机视角，生成半分辨率的间接光照结果，然后在全分辨率下，判断当前像素的间接光照是否可以由半分辨率的相邻的四个 sample 进行双线性插值获得。

判断低分辨率 sample 有效的标准为：sample 的法线和当前像素的法线相似度大于某个阈值，且 sample 的世界空间坐标和当前像素对应的世界空间坐标足够接近。

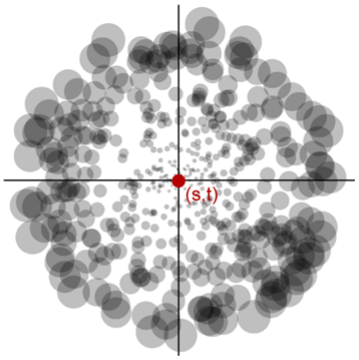

Figure 4: Sampling pattern example. The sample density decreases and the sample weights (visualized by the disk radius) increases with the distance to the center.

图 2: RSM Sampling

2.4 缺点

（1）GI 存在的漏光问题

（2）没有考虑 VPL 和着色点之间的遮挡问题

（3）论文中将所有材质视作 diffuse 材质，实际应用需考虑反射、透明等材质

2.5 参考

RSM

3 Light Propagation Volumes（LPV）

3.1 基本思想

Light Propagation Volumes（LPV）是 CryEngine 3 中使用的全局光照技术，将场景进行体素化表示，利用 RSM 获取 VPL，将 VPL“注入”格子中，在相邻格子之间进行传播，模拟光照多次弹射的过程，迭代一定次数后，得到最终的间接光照结果，并在计算光照时进行采样。

3.2 算法步骤

3.2.1 生成 VPL

利用 RSM 生成 VPL

3.2.2 注入（Injection）

将格子中心视作一个光源，计算每个 VPL 对格子中心点的贡献，并保存在格子中。但是光源在各个方向辐射的能量不是一样的，如果用 CubeMap 进行存储，显存消耗过大。为了减少存储开销，论文中用二阶球谐函数来存储每个格子的光照结果，二阶球谐系数 $\mathbf{c} = (c_0, c_1, c_2, c_3)$。

$$
\begin{aligned}
c_0 &= \frac{1}{2\sqrt{\pi}} \\
c_1 &= -\frac{\sqrt{3}y}{2\sqrt{\pi}} \\
c_2 &= \frac{\sqrt{3}z}{2\sqrt{\pi}} \\
c_3 &= -\frac{\sqrt{3}x}{2\sqrt{\pi}}
\end{aligned}
$$

（1）根据 VPL 的位置信息，和整个 volume 的尺寸，获取 VPL 所在 voxel 的索引以及对应的 voxel 的中心。

（2）以 VPL 中心和格子中心连接的向量作为输入，计算球谐系数。

（3）根据以下公式计算 R、G、B 三个通道的贡献

$$
\begin{pmatrix} \mathbf{c_r} \\ \mathbf{c_g} \\ \mathbf{c_b} \end{pmatrix} = \mathbf{c}^T(\mathbf{I_L}\mathbf{A_S}I_S W_S)
$$

其中 $\mathbf{I_L}$ 为直接光强度，$\mathbf{A_S}$ 为 VPL 对应的 surfel 表面的 albedo；$I_S = (\mathbf{n_s}, \mathbf{l})$，根据法线和相对位置，计算贡献；$W_S$ 为每个 surfel 对应的权重，论文中为落在当前 cell 中的 VPL 的数量和整个 RSM 的 VPL 数量的比值。

3.2.3 传播（Propagation）

如图3所示，每个 voxel 向它相邻的六个 voxel 进行 radiance 传播，达到一定次数后，得到场景的间接光照结果。

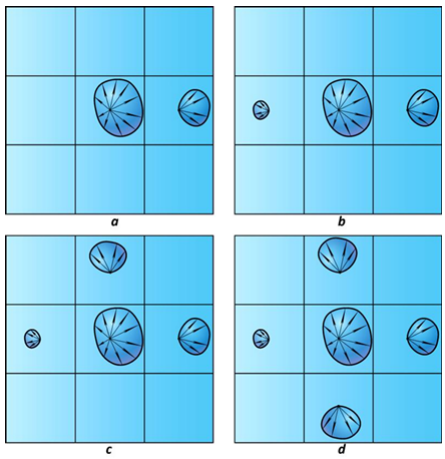

Figure 4. *Radiance propagation iteration*

图 3: Propagation

如图4所示，source cell 将能量传播给 destination cell 的五个面（不包含相邻的那个面），然后以这个五个面作为光源，计算它们对 destination cell 中心点的贡献，与其本身的球谐系数相结合。

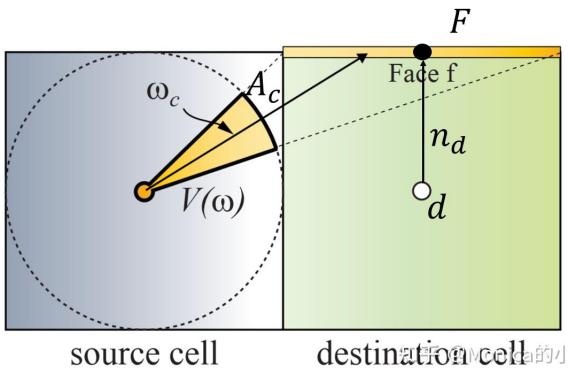

图 4: Propagation Rule

3.3 优化

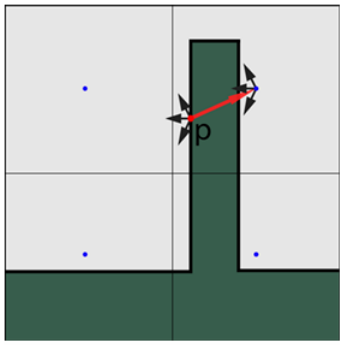

Figure 8. *Light bleeding through a thin double-sided object caused by sparse spatial approximation*

图 5: Light Bleeding

如图5所示，点 p 如果注入到图中对应的 voxel，会导致漏光现象。文中给出的解决方案是，将 VPL 沿着其法线方向或者光源方向移动一定距离（最大为半个 voxel 边长），这样将 VPL 移动到相邻 voxel 中，从而避免漏光。

3.4 缺点

1. voxel 的存储会占用大量显存，voxel 的表示精度对结果有较大影响。

2. 论文中提到只应用于平行光（如太阳），拓展到其他光源会使算法复杂度更高

3.5 参考

LPV

Cascaded Light Propagation Volumes for Real-Time Indirect Illumination

UE4.27 LPV

4 Screen Space Global Illumination（SSGI）

4.1 基本思路

应用屏幕空间的 depth buffer 和 color buffer 计算漫反射的间接光

4.2 算法实现

此处以 Unity 的 HDRP 管线中实现的 SSGI 为例，其分为以下四种模式:

（1）Off （2）ScreenSpace （3）RayTraced （4）Mixed

HDRP 的 SSGI 全部由 Compute Shader 完成，Frame Debugger 中看到的执行流程如下所示:

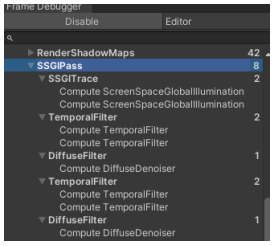

图 6: HDRP SSGI

4.2.1 SSGITrace

（1）第一个步骤为 TraceGlobalIllumination，每个 pixel 对应固定数量的采样方向，对每个采样方向判断碰撞，如果 hit，将对应点的 ndc 坐标存储到纹理中

（1.1）获取 depth 和 normal，根据生成的随机数在法线半球内采样前进方向

（1.2）对光线起始点的世界坐标添加基于 Normal 的 Bias，如下所示

```
1    // Unity 源码
2    // Apply normal bias with the magnitude dependent on the distance from the camera.
3    // Unfortunately, we only have access to the shading normal, which is less than ideal
4    posInput.positionWS  = camPosWS +
5        (posInput.positionWS - camPosWS) *
```

```
6         (1 - 0.001 * rcp(max(dot(normalData.normalWS, viewWS), FLT_EPS)));
```

注：

（a）此处 viewWS 是在世界空间的观察方向，该操作相当于将像素点对应的世界坐标，向相机位置进行偏移，偏移的幅度与像素法线和观察方向的点乘结果有关

（b）此处的法线是着色法线，可能包含法线贴图的结果；可考虑用深度信息重建法线，但这会带来额外的性能开销

（1.3）Ray Marching，记录碰撞点的 NDC 信息，屏幕空间 uv 的信息，对应像素中心的点（Ray Marching 具体步骤见 RayMarch.pdf 文档）

（2）第二个步骤为 ReprojectionGlobalIllumination

（2.1）采样当前像素对应的深度和法线；获取 hit point 的 ndc 和 depth

（2.2）根据 hit point 获取对应的 motion vector，计算上一帧的 hit point 的 ndc 和 depth

如果 ndc 超出取值范围，视为无效；如果上一帧和当前帧的 depth 差值过大，则视为无效

（2.3）计算 GI 结果并存储

（a）有效，利用 hit point 在上一帧的位置采样 color

（b）无效，但是使用 Probe Volume，利用当前像素的位置信息和法线信息对 diffuse 的烘焙结果进行采样

（c）无效，如果有使用反射探针和环境光探针，对它们的结果进行加权计算，获取 color

4.2.2 TemporalFilter

（1）第一个步骤为 TemporalAccumulationColor

根据时序信息对历史结果和当前结果进行加权混合，此处记录下加权因子的计算规则

```
// Accumulation factor
accumulationFactor = sampleCount >= 8.0 ? 0.93 : (sampleCount / (sampleCount + 1.0));
// Update the sample count
sampleCount = min(sampleCount + 1.0, 8.0);

// output
result = color * (1.0 - accumulationFactor) + history.xyz * accumulationFactor
```

（2）第二个步骤为 CopyHistory

4.2.3 DiffuseFilter

此处使用的是 Bilateral Filter，但是与一般的 Bilateral Filter 不同，以代码作为示例分析

```
// 在法向半球内进行采样
float2 newSample = _PointDistribution[sampleIndex + sampleOffset] * denoisingRadius;

// 变换到 clip space
float3 wsPos = center.position + localToWorld[0] * newSample.x
    + localToWorld[1] * newSample.y;
float4 hClip = TransformWorldToHClip(wsPos);
hClip.xyz /= hClip.w;

// 样本半径
float r = length(newSample);

// 计算高斯核的方差
const float sigma = 0.9 * denoisingRadius;

// 计算权重，中心点权重为 1.0
// 权重分为两部分：
// 1. 高斯核，均值为样本半径，方差为固定的 sigma
// 2. 双边过滤权重
const float w = r > 0.001f ?
    gaussian(r, sigma) * ComputeBilateralWeight(center, tapData) : 1.0;
```

双边过滤的权重为为三项乘积，分别为 depth weight、normal weight 和 plane weight

```
float ComputeBilateralWeight(BilateralData center, BilateralData tap)
{
    float depthWeight    = 1.0;
    float normalWeight   = 1.0;
    float planeWeight    = 1.0;

    if (DEPTH_WEIGHT > 0.0)
    {
```

```
9           depthWeight = max(0.0, 1.0 - abs(tap.z01 - center.z01) * DEPTH_WEIGHT);
10      }
11
12      if (NORMAL_WEIGHT > 0.0)
13      {
14          const float normalCloseness = sqr(sqr(max(0.0, dot(tap.normal, center.normal))));
15          const float normalError = 1.0 - normalCloseness;
16          normalWeight = max(0.0, (1.0 - normalError * NORMAL_WEIGHT));
17      }
18
19      if (PLANE_WEIGHT > 0.0)
20      {
21          // Change in position in camera space
22          const float3 dq = center.position - tap.position;
23
24          // How far away is this point from the original sample
25          // in camera space? (Max value is unbounded)
26          const float distance2 = dot(dq, dq);
27
28          // How far off the expected plane (on the perpendicular)
29          // is this point? Max value is unbounded.
30          const float planeError =
31              max(abs(dot(dq, tap.normal)), abs(dot(dq, center.normal)));
32
33          planeWeight = (distance2 < 0.0001) ? 1.0 :
34          pow(max(0.0, 1.0 - 2.0 * PLANE_WEIGHT * planeError / sqrt(distance2)), 2.0);
35      }
36
37      return depthWeight * normalWeight * planeWeight;
38  }
```

4.2.4 TemporalFilter

与前一个 TemporalFilter 相同

4.2.5 DiffuseFilter

与前一个 DiffuseFilter 相同

4.3 参考

Screen Space Global Illumination