

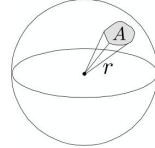
渲染

张淦淦

2022 年 6 月 18 日

目录

I 实时渲染的模型	2	4.2.3 Screen Space Directional Occlusion(SSDO)	12
1 PBR(Physical Based Rendering)	2	4.2.4 Screen Space Reflection(SSR)	13
1.1 辐射度量学基础	2	4.3 Radiosity算法	13
1.2 BRDF	3	4.4 光线追踪算法	13
1.3 反射方程与渲染方程	3	4.4.1 基本光线追踪算法	13
1.4 光与材质交互的物理规律	3	4.4.2 基于降噪技术的实时光线追踪	14
1.5 微表面模型	4		
2 局部光照	5	II 实时渲染的实现	15
2.1 点光源,聚光灯,方向光,面光源的渲染	5	5 渲染管线	15
2.2 环境光渲染(Environment lighting)	7	5.1 Application Stage	15
2.2.1 Light Map	7	5.2 顶点着色器	15
2.2.2 lightProbe	7	5.3 可选着色器	17
2.2.3 Image-Based Lighting(IBL)	7	5.4 光栅化(Rasterization)	17
2.2.4 Environment Mapping	8	5.5 Pixel Processing	18
		5.6 小结	18
3 阴影	8	6 常用空间数据结构	18
3.1 Shadow Mapping	8	6.1 基于空间细分的数据结构	18
3.2 软阴影	9	6.1.1 BSP树	19
3.2.1 PCSS	9	6.1.2 KD树	19
3.2.2 Variance Soft Shadow Mapping	10	6.1.3 八叉树	19
3.3 Moment Shadow Mapping	10	6.2 层次包围盒(BVH)	19
3.3.1 Distance Field Soft Shadows	10	6.2.1 在GPU上实现BVH	20
4 全局光照	10	6.3 场景图	21
4.1 3D Space	10	7 剔除技术	21
4.1.1 Reflective Shadow Map(RSM)算法	10	8 层次细节(LOD)	22
4.1.2 Light Propagation Volumes(LPV)算法	11	9 纹理	22
4.1.3 Voxel Global Illumination(VXGI)算法	12	9.1 纹理中的插值算法	22
4.2 Screen Space	12	9.2 纹理中的降采样方法	22
4.2.1 Screen Space Ambient Occlusion(SSAO)	12	9.3 Normal Mapping	23
4.2.2 Horizon Based Ambient Occlusion(HBAO)	12		

10 实际渲染管线中的技术	24	辐射强度(Radiant Intensity),点光源所发射光的每单位立体角内功率(power).
10.1 高效Shading算法	24	$I(\omega) = \frac{d\Phi}{d\omega} [\frac{W}{sr}] [\frac{lm}{sr}] = cd = candela]$
10.2 抗锯齿算法	24	对于各向同性点光源 $I = \Phi/4\pi$
10.3 图像后处理	25	立体角: 球面上面积与对应半径平方的比值,球为 $4\pi \text{ steradians}$
10.4 Tone Mapping	25	$\Omega = \frac{A}{r^2}$
10.5 渲染架构	25	
III 数学	25	
11 RTR中的一些基础知识与推导	25	
11.1 Mente Carlo积分	25	
11.2 Rodrigues' Rotation Formula推导	25	
11.3 重心坐标	26	
11.4 Perspective Correct Interpolation	27	
11.5 Surface Normal Transform	27	
11.6 Perspective Projection推导	28	
11.7 RTR约等式	28	
12 几何求交	28	
12.1 射线求交	28	
12.2 面求相交测试	30	
12.3 体的相交测试	31	
12.4 View Frustum相交测试	31	
13 球谐函数		
14 随机采样		
14.1 采样方法	32	
14.2 在流形上均匀采样的通用做法	33	

Part I

实时渲染的模型

1 PBR(Physical Based Rendering)

1.1 辐射度量学基础

辐射能量(Radiant Energy),指电磁波能量

$$Q[J = Joule]$$

辐射通量(Radiant Flux(Power)),是每单位时间内发射,反射,传播或接受的能量.

$$\Phi = \frac{dQ}{dt} [W = Watt] [lm = lumen]$$

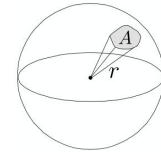
辐射强度(Radiant Intensity),点光源所发射光的每单位立体角内功率(power).

$$I(\omega) = \frac{d\Phi}{d\omega} [\frac{W}{sr}] [\frac{lm}{sr}] = cd = candela]$$

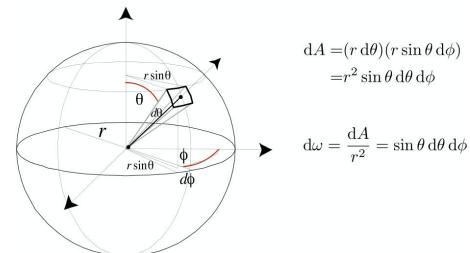
对于各向同性点光源 $I = \Phi/4\pi$

立体角: 球面上面积与对应半径平方的比值,球为 $4\pi \text{ steradians}$

$$\Omega = \frac{A}{r^2}$$



微分立体角



辐射照度(Irradiance),每单位面积(面与光线垂直)上接收到的辐射功率.

$$E(x) = \frac{d\Phi(x)}{dA} [\frac{W}{m^2}] [\frac{lm}{m^2}] = lux$$

辐射度量(Radiance),指表面每单位投影面积,单位立体角接受发射,反射,传播或接受的辐射功率.



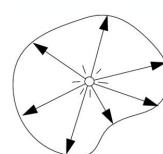
$$L(p, \omega) = \frac{d^2\Phi(p, \omega)}{d\omega dA \cos \theta} [\frac{W}{sr m^2}] [\frac{cd}{m^2}] = \frac{lm}{sr m^2} = nit$$

入射辐射度量(Incident Radiance),每单位立体角到达表面某点的辐射照度.

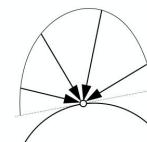
$$L_i(p, \omega) = \frac{dE(p)}{d\omega \cos \theta}$$

出射辐射度量(Exiting Radiance),每单位投影面积辐射强度.

$$L_r(p, \omega) = \frac{dI(p, \omega)}{dA \cos \theta}$$



Light Emitted From A Source
"Radiant Intensity"



Light Falling On A Surface
"Irradiance"

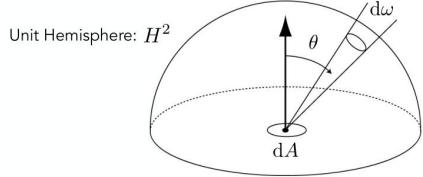


Light Traveling Along A Ray
"Radiance"

通过Radiance求得Irradiance.

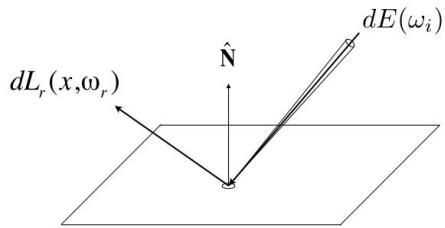
$$dE(p, \omega) = L_i(p, \omega) \cos\theta d\omega$$

$$E(p) = \int_{H^2} L_i(p, \omega) \cos\theta d\omega$$



1.2 BRDF

双向反射分布函数(Bidirectional Reflectance Distribution Function),描述了入射光线能量是如何被反射到各个方向的.



$$f_r(\omega_i, \omega_r) = \frac{dL_r(\omega_r)}{dE(\omega_i)} = \frac{dL_r(\omega_r)}{L_i(\omega_i) \cos\theta_i d\omega_i} [\text{sr}]$$

1.3 反射方程与渲染方程

反射方程:

$$L_r(p, \omega_r) = \int_{H^2} f_r(p, \omega_i, \omega_r) L_i(p, \omega_i) \cos\theta_i d\omega_i$$

渲染方程:

$$L_r(p, \omega_r) =$$

$$L_e(p, \omega_r) + \int_{H^2} f_r(p, \omega_i, \omega_r) L_r(p'(p, \omega_i), -\omega_i) (n \cdot \omega_i) d\omega_i$$

其中, $p'(p, \omega_i)$ 指空间中与 p 连线方向与 ω_i 一致的点.

1.4 光与材质交互的物理规律

漫反射 物体材质由BRDF反映,如漫反射材质(Lambertian材质),假设任意角度入射辐照度量相同,且等于任意角度出射辐射度量,即 $L_r(\omega_r) = L_i$,根据渲染方程

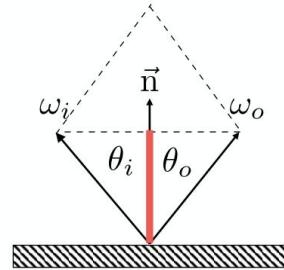
$$\begin{aligned} L_r(\omega_r) &= \int_{H^2} f_r L_i(\omega_i) \cos\theta_i d\omega_i \\ &= f_r L_i \int_{H^2} \cos\theta_i d\omega_i \\ &= \pi f_r L_i \end{aligned}$$

所以有,

$$f_r = \frac{1}{\pi}$$

一般物理世界,物体吸收光的能量,所以可以定义反射系数 $\rho \in [0, 1]$, 则 $f_r = \rho/\pi$

反射光的角度

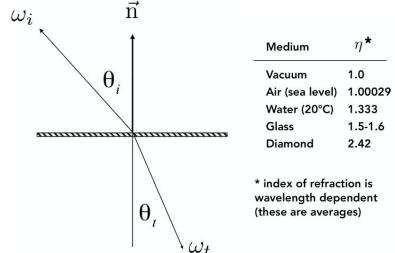


$$\theta = \theta_i = \theta_o$$

$$\omega_i + \omega_o = 2 \cos\theta n = 2(\omega_i \cdot n)n$$

$$\omega_o = -\omega_i + 2(\omega_i \cdot n)n$$

折射光的角度(Snell's Law)



$$\eta_i \sin\theta_i = \eta_t \sin\theta_t$$

$$\cos\theta_t = \sqrt{1 - (\frac{\eta_i}{\eta_t})^2(1 - \cos^2\theta_i)}$$

设 $i = -w_i$ 在平面上投影为 $i_{||}, i_{\perp} = i - i_{||}$, 同理, $t = w_t$ 在平面上投影为 $t_{||}, t_{\perp} = t - t_{||}$

$$r_{||} = \frac{\sin\theta_r}{\sin\theta_i} i_{||} = \frac{n_1}{n_2} i_{||} = \frac{n_1}{n_2} (i + \cos\theta_i n)$$

$$|r_{\perp}| = \sqrt{|r|^2 - |r_{||}|^2} = \sqrt{1 - \sin^2\theta_t}$$

$$r_{\perp} = -|r_{\perp}|n = -\sqrt{1 - \sin^2\theta_t}n = -\sqrt{1 - (\frac{\eta_i}{\eta_t})^2(1 - \cos^2\theta_i)}n$$

$$\begin{aligned} r &= r_{||} + r_{\perp} \\ &= \frac{n_1}{n_2} (i + \cos\theta_i n) - \sqrt{1 - (\frac{\eta_i}{\eta_t})^2(1 - \cos^2\theta_i)}n \\ &= \frac{n_1}{n_2} i + (\frac{n_1}{n_2} \cos\theta_i - \sqrt{1 - (\frac{\eta_i}{\eta_t})^2(1 - \cos^2\theta_i)})n \end{aligned}$$

折射光与反射光能量分布(Fresnel Reflection/Term)

菲涅尔项解释了具体有多少能量被折射,多少能量被反射

$$R_s = \left| \frac{n_1 \cos \theta_i - n_2 \cos \theta_t}{n_1 \cos \theta_i + n_2 \cos \theta_t} \right|^2$$

$$R_p = \left| \frac{n_1 \cos \theta_t - n_2 \cos \theta_i}{n_1 \cos \theta_t + n_2 \cos \theta_i} \right|^2$$

$$R_{eff} = \frac{1}{2}(R_s + R_p)$$

$R_{eff} \in [0, 1]$ 是反射的能量比例.按定义算太复杂,一般采用Schlick's近似

$$R(\theta) = R_0 + (1 - R_0)(1 - \cos \theta)^5$$

$$R_0 = \left(\frac{n_1 - n_2}{n_1 + n_2} \right)^2$$

1.5 微表面模型

Phong和Blinn Phong等模型都是基于观察提出的经验模型以达到看起来一致,而PBR是通过光物理性质推导出来的一类shading model,PBR不一定完全物理正确,在有些情况进行了近似.

微表面模型等是对材质物理属性的更细致精确一些的描述方法

$$L_r(p, \omega_r) = L_e(p, \omega_r) + \int_{\Omega} \left(k_d \frac{c}{\pi} + k_s \frac{F(i, h)G(i, o, h)D(h)}{4(n \cdot i)(n \cdot o)} \right) L_i(p, \omega_i)(\omega_i, n) d\omega_i$$

可以理解为

输出颜色 =

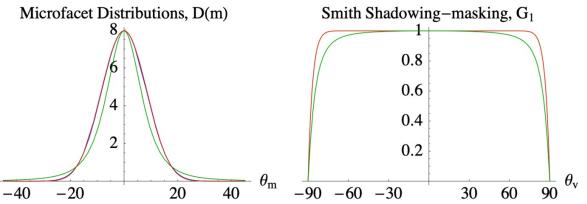
发射光颜色
 $+ \int_{\Omega} (\text{漫反射比例} \frac{\text{纹理颜色}}{\pi})$
 $+ \text{镜面反射比例} \frac{\text{菲涅尔效应} \times \text{几何遮蔽} \times \text{法向分布}}{4(\text{viewDir} \cdot \text{normal})(\text{lightDir} \cdot \text{normal})}$
光源颜色($\text{lightDir} \cdot \text{normal}$) $d\omega_i$

设,其中 h 表示的是half vector

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

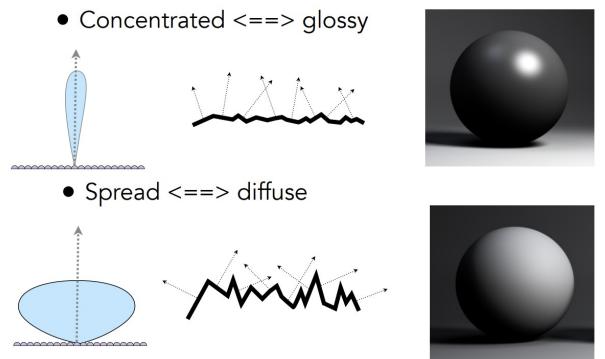
- 菲涅尔项(Fresnel Term), $F(i, h)$ 用一般用Schlick's近似
- 几何遮蔽项目 $G(i, o, h)$,解决微表面之间自遮挡问题.提供由于遮挡产生的变暗(grazing angle处最暗);如果没有 G 项,在grazing angle处,微表面模型分母为接近0,此时会得到很亮的白边.

1. Smith shadowing-masking.假设 $G(i, o, h) \approx G_1(i, h)G_1(o, h)$



G_1 与NDF项有关.

- 法向分布项(Normal Distribution Function,NDF), $D(h)$:

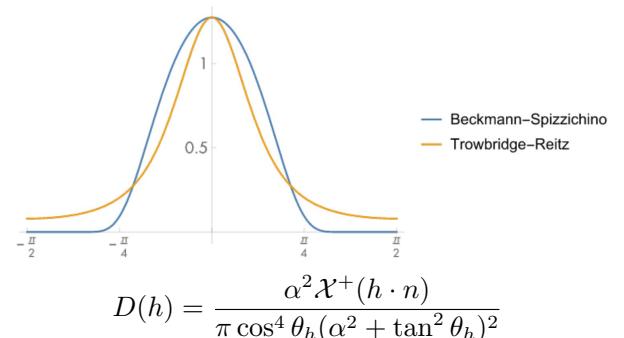


1. Beckmann distribution

$$D(h) = \frac{e^{\frac{\tan^2 \theta_h}{\alpha^2}}}{\pi \alpha^2 \cos^4 \theta_h}$$

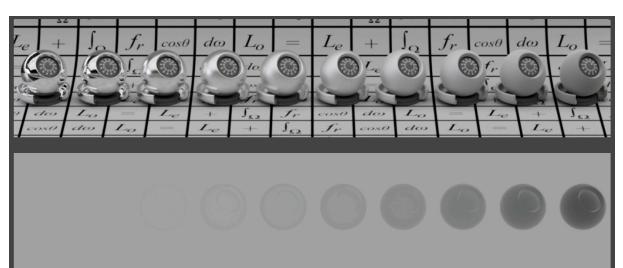
α 影响diffuse和glossy(胖瘦), θ_h 是 h 与 n 的夹角

2. GGX(或Trowbridge-Reitz),明显特点是long tail,在高光周围出现光晕.而Beckmann高光周围比较sharp.



3. Extending GGX, γ 参数控制long tail,当 $\gamma = 2$ 等价于GGX, γ 很大时接近Beckmann.

根据以上得到的微表面模型有一点问题,当表面粗糙度越高时,损失能量越多.



因为表面越粗糙,光线在表面多次弹射的概率就越大,但是微表面模型只考虑了一次弹射,所以需要补偿,在离线渲染领域有精确解,在实时渲染中使用Kulla-Conty Approximation.

首先,在不考虑颜色的情况下,从表面一次弹射出来的能量为

$$E(\mu_o) = \int_0^{2\pi} \int_0^1 f(\mu_o, \mu_i, \phi) \mu_i d\mu_i d\phi, \quad \mu = \sin \theta$$

所以,多次弹射的总能量为 $1 - E(\mu_o)$,设补偿BRDF为

$$f_{ms}(\mu_o, \mu_i) = c(1 - E(\mu_i))(1 - E(\mu_o))$$

其中 c 为一个常数

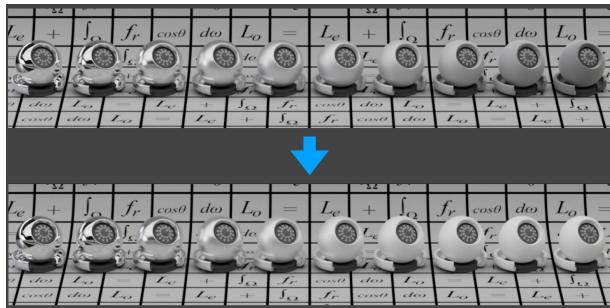
$$\int_0^{2\pi} \int_0^1 f_{ms}(\mu_o, \mu_i) \mu_i d\mu_i d\phi = 1 - E(\mu_o)$$

从如上等式推出 f_{ms} 的形式

$$f_{ms}(\mu_o, \mu_i) = \frac{(1 - E(\mu_o))(1 - E(\mu_i))}{\pi(1 - E_{avg})}, \quad E_{avg} = 2 \int_0^1 E(\mu) \mu d\mu$$

$E_{avg} = 2 \int_0^1 E(\mu) \mu d\mu$ 通过预算算打表(依赖参数 μ ,粗糙度 α),最终使用时将补充BRDF和实际BRDF相加,即

$$funcor(\mu_o, \mu_i, \phi) = f(\mu_o, \mu_i, \phi) + f_{ms}(\mu_o, \mu_i)$$



考虑由于颜色吸收造成的能力损失,定义平均Fresnel(多少能量被反射)

$$F_{avg} = \frac{\int_0^1 F(\mu) \mu d\mu}{\int_0^1 \mu d\mu} = 2 \int_0^1 F(\mu) \mu d\mu$$

能够直接看到的能量

$$F_{avg} E_{avg}$$

一次弹射后能看到的能量

$$F_{avg}(1 - E_{avg}) F_{avg} E_{avg}$$

k 次弹射后能看到的能量

$$F_{avg}^k (1 - E_{avg})^k F_{avg} E_{avg}$$

求和得到颜色项

$$\sum_{k=0}^{\infty} F_{avg}^k (1 - E_{avg})^k F_{avg} E_{avg} = \frac{F_{avg} E_{avg}}{1 - F_{avg}(1 - E_{avg})}$$

将这个颜色项乘到uncolored additional BRDF项上,即

$$f_{cor}(\mu_o, \mu_i, \phi) = \frac{F_{avg} E_{avg}}{1 - F_{avg}(1 - E_{avg})} funcor(\mu_o, \mu_i, \phi)$$

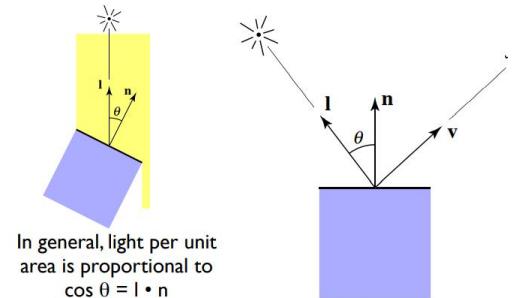
Disney Principlee BRDF

2 局部光照

局部光照只研究直接光照与着色点局部表面的作用,不考虑遮挡产生的阴影,也不考虑间接光照.

2.1 点光源,聚光灯,方向光,面光源的渲染

Lambert漫反射模型



$$L_d = k_d (I/r^2) \max(0, \mathbf{n} \cdot \mathbf{l})$$

L_d 表示漫反射光的辐射度量, k_d 表示漫反射系数, r 表示点光源到shading point的距离, $\max(0, \mathbf{n} \cdot \mathbf{l})$ 表示shading point接受的能量.

半Lambert模型 将shading point能接受的能量映射到了 $[0, 1]$ 范围内

$$L_d = k_d (I/r^2) (\mathbf{n} \cdot \mathbf{l} * 0.5 + 0.6)$$

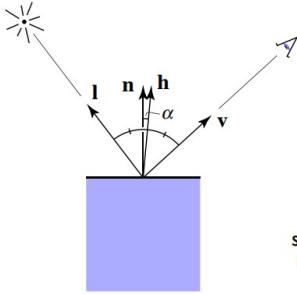
Phong模型 存在问题,当视线 v 与反射光线 R 夹角大于90度时,出现截断,而如果表面粗糙镜面反射分量此时还有应该贡献. Blinn Phong克服了这个问题,此外, Blinn Phong不需要计算反射向量,速度更快,所以一般使用Blinn Phong.

$$L_d = k_d (I/r^2) \max(0, \mathbf{n} \cdot \mathbf{l})$$

$$L_s = k_s (I/r^2) \max(0, \mathbf{v} \cdot \mathbf{R})^p$$

$$L_a = k_a I_a$$

Blinn Phong光照模型



$$L_d = k_d(I/r^2) \max(0, n \cdot l)$$

镜面反射辐射度量

$$h = bisector(v, l) = \frac{v + l}{\|v + l\|}$$

$$L_s = k_s(I/r^2) \max(0, n \cdot h)^p$$

用 $n \cdot h$ 度量是否与反射 R 靠近, 指数 p 用来修正度量指标, 经验上取 $100 \sim 200$.

环境光照辐射度量, 其中 I_a 是一个常数

$$L_a = k_a I_a$$

Blinn Phong 模型

$$L = L_a + L_d + L_s$$

$$= k_a I_a + k_d(I/r^2) \max(0, n \cdot l) + k_s(I/r^2) \max(0, n \cdot h)^p$$

顶点着色器

```
#version 330 core
layout (location = 0) in vec4 pos;
layout (location = 1) in vec2 uv;
layout (location = 2) in vec3 norm;
out VS_OUT {
    vec4 frag_pos;
    vec2 frag_uv;
    vec3 frag_norm;
} vs_out;
uniform mat4 Mm, Mv, Mp;
void main() {
    gl_Position = Mp * Mv * Mm * pos;
    vs_out.frag_pos = Mm * pos;
    vs_out.frag_uv = uv;
    vs_out.frag_norm = mat3(transpose(inverse(Mm))) * norm;
}
```

片段着色器

```
#version 330 core
in VS_OUT {
    in vec4 frag_pos;
    in vec2 frag_uv;
    in vec3 frag_norm;
} fs_in;
out vec4 color;
uniform sampler2D tex0;
uniform vec3 light_ambient, light_pos, light_color, camera_pos;
void main() {
```

```
    vec3 obj_color = texture(tex0, vec2(fs_in.frag_uv.x, 1-fs_in.frag_uv.y)).rgb;
    vec3 ambient = light_ambient;
    vec3 light_dir = normalize(light_pos - fs_in.frag_pos.xyz);
    vec3 normal = normalize(fs_in.frag_norm);
    float diff_factor = max(dot(light_dir, normal), 0.0);
    vec3 diffuse = diff_factor * light_color;
    vec3 view_dir = normalize(camera_pos - fs_in.frag_pos.xyz);
    vec3 half_dir = normalize(light_dir + view_dir);
    float spec_factor = pow(max(dot(half_dir, normal), 0.0), 32.0); // 32.0为镜面高光系数
    vec3 specular = spec_factor * light_color;
    vec3 res_color = (ambient + diffuse + specular) * obj_color;
    color = vec4(res_color, 1.0f);
}
```

微表面模型在多边形面光源下的渲染 D_o 是原始分布, ω_o 是原始分布的中心轴.

$$D_o(\omega_o = (x, y, z)) = \frac{1}{\pi} \max(0, z)$$

将中心轴 ω_o 通过 $M \in \mathbb{R}^{3 \times 3}$ 变换得到 ω ,

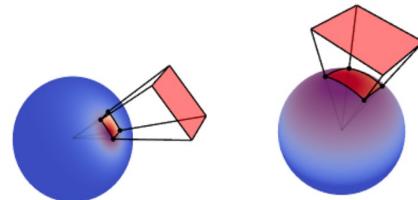
$$\omega = \frac{M\omega_o}{\|M\omega_o\|}, \quad \omega_o = \frac{M^{-1}\omega}{\|M^{-1}\omega\|}$$

$$D(\omega) = D_o(\omega) \frac{\partial \omega_o}{\partial \omega} = D_o\left(\frac{M^{-1}\omega}{\|M^{-1}\omega\|}\right) \frac{|M^{-1}|}{\|M^{-1}\omega\|^3}$$

性质:

$$\int_P D(\omega) d\omega = \int_{P_o} D_o(\omega_o) d\omega_o = E(P_o)$$

其中, P 是多边形光源, $P_o = M^{-1}P$ 是多边形通过逆仿射变换到原始分布下的结果, 如下图, 两者积分是相等的



因为 D_o 是 clamped cosine distribution, 如果 polygon 在上半球区间, 则其在 polygon 上积分有闭式解

$$E(p_1, \dots, p_n) = \frac{1}{2\pi} \sum_{i=1}^n \cos(p_i \cdot p_j) \frac{p_i \times p_j}{\|p_i \times p_j\|} \cdot \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$$

$$j = (i+1) \mod n$$

LTC核心思想是用 $D(\omega)$ 近似 $f_r(p, \omega, \omega_r, \alpha) \cos \theta_i$, 然后逆变换到 $D_o(\omega)$ 求多边形积分. 先通过预算算求得 $M_{p, \omega_r, \alpha}$

$$M_{p, \omega_r, \alpha} = \underset{M}{argmin} \|D_o\left(\frac{M^{-1}\omega}{\|M^{-1}\omega\|}\right)\frac{|M^{-1}|}{\|M^{-1}\omega\|^3} - f_r(p, \omega, \omega_r, \alpha) \cos \theta_i\|$$

若光源分布恒定, 即 $L_i(\omega) = L$, 则对于 shading point, p , 其 illumination 到 Shading without shadow, 只考虑反射方程.

$$\begin{aligned} L_r(p, \omega_r) &= \int_P f_r(p, \omega_i, \omega_r, \alpha) L_i(p, \omega_i) \cos \theta_i d\omega_i \\ &= \int_P D_{p, \omega_r, \alpha}(\omega_i) L_i(p, \omega_i) d\omega_i \\ &= L_i \int_P D_{p, \omega_r, \alpha}(\omega_i) d\omega_i \\ &= L_i \int_{M_{p, \omega_r, \alpha}^{-1} P} D_o(\omega_i) d\omega_i \\ &= L_i E(M_{p, \omega_r, \alpha}^{-1} P) \end{aligned}$$

若光源是有纹理的,

$$\begin{aligned} L_r(p, \omega_r) &= \int_P D_{p, \omega_r, \alpha}(\omega_i) L_i(p, \omega_i) d\omega_i \\ &= \int_P D_{p, \omega_r, \alpha}(\omega_i) d\omega_i \frac{\int_P D_{p, \omega_r, \alpha}(\omega_i) L_i(p, \omega_i) d\omega_i}{\int_P D_{p, \omega_r, \alpha}(\omega_i) d\omega_i} \\ &= E(M_{p, \omega_r, \alpha}^{-1} P) \frac{\int_P D_{p, \omega_r, \alpha}(\omega_i) L_i(p, \omega_i) d\omega_i}{\int_P D_{p, \omega_r, \alpha}(\omega_i) d\omega_i} \end{aligned}$$

对于后半部分, 可以视作是光照纹理 $L_i(p, \omega_i)$ 通过滤波器

$$F(\omega_i) = \frac{D_{p, \omega_r, \alpha}(\omega_i)}{\int_P D_{p, \omega_r, \alpha}(\omega_i) d\omega_i}$$

后的结果, 将 F 用 Gaussian 滤波器近似, 即先将纹理通过高斯滤波器处理, 然后, 对于每个 shading point,

$$\frac{\int_P D_{p, \omega_r, \alpha}(\omega_i) L_i(p, \omega_i) d\omega_i}{\int_P D_{p, \omega_r, \alpha}(\omega_i) d\omega_i} = \frac{\int_{P_o} D_o(\omega_i) L_o(p, \omega_i) d\omega_i}{\int_{P_o} D_o(\omega_i) d\omega_i}$$

具体做法, 将光照分布纹理 $L_i(\omega)$ 高斯滤波处理, 然后在 o 坐标系, 从 shading point, p 开始, 沿着 ω_i 发出光线, 求与 P_o 交点, 得到对应滤波后的纹理.

着色频率

- Flat shading : 每个三角形只有一个法向
- Gouraud shading : 每个顶点有一个法向量, 周围三角形法向的平均, 发生在 Vertex 着色过程.
- Phong shading : 在每个像素上计算 shading model, 用 Barycentric Interpolation. 发生在 Fragment 着色过程.

2.2 环境光渲染(Environment lighting)

2.2.1 Light Map

2.2.2 Light Probe

2.2.3 Image-Based Lighting(IBL)

讨论如何由环境光照(Environment Lighting)得

到 Shading without shadow, 只考虑反射方程.

$$\begin{aligned} L_r(p, \omega_r) &= \int_{H^2} f_r(p, \omega_i, \omega_r) L_i(p, \omega_i) (n \cdot \omega_i) d\omega_i \\ &= \int_{H^2} f_r(p, \omega_i, \omega_r) L_i(p, \omega_i) \cos \theta_i V(n, \omega_i) d\omega_i \end{aligned}$$

不考虑可见性(遮挡).

$$L_r(p, \omega_r) = \int_{H^2} f_r(p, \omega_i, \omega_r) L_i(p, \omega_i) \cos \theta_i d\omega_i$$

这个反射方程求解理论上可以使用蒙特卡洛积分, 即, 对于每一个 shade point, p . 对 p 半球空间内采样(求多个 ω_r), 但是这在实时渲染中是不可取的.

对于某个材质,

- 如果其 BRDF 主要是 glossy 的, 只有少部分从 ω_i 射入的光线能从 ω_r 射出来, 对应 f_r 的支撑集很小.
- 如果 BRDF 是 diffuse 的, 那么说明 f_r 很光滑.

所以应用“RTR 约等式”简化反射方程

$$L_r(p, \omega_r) \approx \underbrace{\frac{\int_{\Omega_{f_r}} L_i(p, \omega_i) d\omega_i}{\int_{\Omega_{f_r}} d\omega_i}}_{stage1} \underbrace{\int_{H^2} f_r(p, \omega_i, \omega_r) \cos \theta_i d\omega_i}_{stage2}$$

stage 1 光照信息存在球上 $L_{env}(\omega_i)$, 对 $L_{env}(\omega_i)$ 以一系列立体角半径为 r_1, r_2, \dots, r_n 做低通滤波(具体滤波方法?)得到 $L_{env}(r_1, \omega_i), \dots, L_{env}(r_n, \omega_i)$, 插值得到插值函数 $L_{env}(r, \omega_i)$

如果出射光方向为 ω_o , 根据 BRDF, 射入射光方向为 $\omega_i \in \Omega_{f_r}$ 时 $f_r(p, \omega_i, \omega_o) \neq 0$, 则 p 点光辐射强度为 $L_i(\int_{\Omega_{f_r}} d\omega_i, \omega_i)$

stage 2 假设 BRDF 用微表面模型, 且不考虑几何遮蔽 $G(\omega_i, \omega_r, h)$

$$f_r(p, \omega_i, \omega_r) = \frac{F(\omega_i, h) D(h)}{4(n \cdot \omega_i)(n \cdot \omega_r)} = \frac{F(\omega_i, h) D(h)}{4 \cos \theta_i \cos \theta_r}$$

基本思想也是: 将积分问题转换成预计算查表, 直接建表维度太高, 所以做一个数学上处理, 为表面模型中 Fresnel 项用 Schlick 近似.

$$\int_{H^2} f_r(p, \omega_i, \omega_r) \cos \theta_i d\omega_i \approx$$

$$\begin{aligned} & R_0 \int_{H^2} \frac{f_r(p, \omega_i, \omega_r)}{F(\omega_i, h)} (1 - (1 - \cos \theta_i)^5) \cos \theta_i d\omega_i \\ & + \int_{H^2} \frac{f_r(p, \omega_i, \omega_r)}{F(\omega_i, h)} (1 - \cos \theta_i)^5 \cos \theta_i d\omega_i \\ & = R_0 \int_{H^2} \frac{D(h)}{4 \cos \theta_r} (1 - (1 - \cos \theta_i)^5) d\omega_i \\ & + \int_{H^2} \frac{D(h)}{4 \cos \theta_r} (1 - \cos \theta_i)^5 d\omega_i \end{aligned}$$

注意,实时渲染中入射光与法向量夹角 θ_i ,出射光与法向量夹角 θ_o ,入射光出射光夹角一半与法向夹角 θ_h 可以互换,即

$$\theta_i \approx \theta_o \approx \theta_h$$

若采用Beckmann NDF

$$D(h) \approx \frac{e^{\frac{\tan^2 \theta_i}{\alpha^2}}}{\pi \alpha^2 \cos^4 \theta_i}$$

$$\int_{H^2} f_r(p, \omega_i, \omega_r) \cos \theta_i d\omega_i \approx \text{IntMap}(\alpha, \cos \theta_i)$$

$$\text{IntMap}(\alpha, \cos \theta_i) =$$

$$R_0 \int_{H^2} \frac{e^{\frac{\tan^2 \theta_i}{\alpha^2}}}{4 \cos \theta_r \pi \alpha^2 \cos^4 \theta_i} (1 - (1 - \cos \theta_i)^5) d\omega_i + \int_{H^2} \frac{e^{\frac{\tan^2 \theta_i}{\alpha^2}}}{4 \cos \theta_r \pi \alpha^2 \cos^4 \theta_i} (1 - \cos \theta_i)^5 d\omega_i$$

预先求积分,建立表 $\text{IntMap}(\alpha, \cos \theta_i)$,渲染时直接查表即可

2.2.4 Environment Mapping

是一种利用预计算纹理高效Image-based lighting的技术,解决给定环境光情况下算shading和shadow,给定环境光 $L(\omega_i)$,用基函数(如球谐基)逼近环境光.

$$L(\omega_i) = \sum_k l_k B_k(\omega_i)$$

预计算求系数

$$l_k = \int_{\Omega} L(\omega) B_k(\omega) d\omega$$

对于反射方程

$$L_r(p, \omega_r) = \int_{H^2} \underbrace{L_i(p, \omega_i)}_{\text{lighting}} \underbrace{V(n, \omega_i) f_r(p, \omega_i, \omega_r) \max(0, \omega_i \cdot n)}_{\text{light transport, LT}} d\omega_i$$

如果场景中物体静止,对于每个点light transport不会变化,同样light transport中,视角确定时, ω_r 不变, n 在 p 点不变,唯一变量是 ω_i ,故也可以对Light Transport用基函数逼近,设

$$LT(p, \omega_i, \omega_r) = V(n, \omega_i) f_r(p, \omega_i, \omega_r) \max(0, \omega_i \cdot n) \approx \sum_l t_{pl}(\omega_r) B_l(\omega_i)$$

$$t_{pl}(\omega_r) \approx \sum_m t_{plm} B_m(\omega_r)$$

$$LT(p, \omega_i, \omega_r) \approx \sum_k \sum_m t_{plm} B_m(\omega_r) B_l(\omega_i)$$

求系数 $t_{pl}(\omega_r), t_{plm}$

$$t_{pl}(\omega_r) = \int_{\Omega} LT(p, \omega, \omega_r) B_l(\omega) d\omega$$

$$\begin{aligned} t_{plm} &= \int_{\Omega} t_{pl}(\theta) B_m(\theta) d\theta \\ &= \int_{\Omega} \int_{\Omega} LT(p, \omega, \theta) B_l(\omega) B_m(\theta) d\omega d\theta \end{aligned}$$

对任意一个 p ,给定视线 ω_r ,若已知其 t_{plm} ,则可通过下式求 L_r

$$\begin{aligned} L_r(p, \omega_r) &\approx \int_{H^2} \sum_k l_k B_k(\omega_i) \sum_l \sum_m t_{plm} B_m(\omega_r) B_l(\omega_i) d\omega_i \\ &= \sum_k \sum_l \sum_m l_k t_{plm} B_m(\omega_r) \int_{H^2} B_k(\omega_i) B_l(\omega_i) d\omega_i \\ &= \sum_l l_l t_{plm} B_m(\omega_r) \end{aligned}$$

在场景主要为漫反射的情况下 $f_r(p, \omega_i, \omega_r) = \rho$

$$LT(p, \omega_i) = \rho V(n, \omega_i) \max(0, \omega_i \cdot n) \approx \sum_l t_{pl} B_l(\omega_i)$$

$$\begin{aligned} t_{pl} &= \int_{\Omega} LT(p, \omega) B_l(\omega) d\omega \\ L_r(p, \omega_r) &\approx \int_{H^2} \sum_k l_k B_k(\omega_i) \sum_l t_{pl} B_l(\omega_i) d\omega_i \\ &= \sum_k \sum_l l_k t_{pl} \int_{H^2} B_k(\omega_i) B_l(\omega_i) d\omega_i \\ &= \sum_l l_l t_{pl} \end{aligned}$$

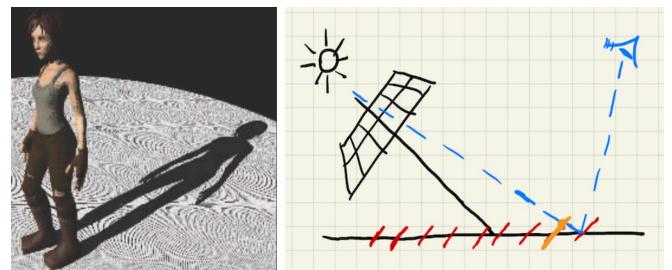
3 阴影

3.1 硬阴影

3.1.1 Shadow Mapping

是一个2-Pass算法,先从light pass生成SM,Camera Pass使用SM.是一个screen space算法.

Shadow Mapping存在的问题,在下图中出现self occlusion问题,本质是精度问题,左图是出现的问题,其原因在右图中描述,light pass中记录SM深度,本质是将场景离散化成如图所示一个个小的斜面,假定各小斜面到light source的距离是一致的.在camera pass中人眼看向场景中某点,该点与光源连线有可能被其他小斜面遮挡导致本该有光点由于自遮挡变得无光.



当光线方向与平面垂直时问题最小,当光线方向与平面接近平行时问题最大.解决方法是设置一个阈值,要求点比SM深度要深过一定阈值才被认为是遮挡而不是自遮挡,阈值要求合理不然会出现误判.

另外一个问题是Aliasing(走样),解决方法是Cascade Shadow Mapping.

从数学角度理解SM 对于渲染方程

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

用RTR约等式

$$L_o(p, \omega_o) = \frac{\int_{\Omega^+} V(p, \omega_i) d\omega_i}{\int_{\Omega^+} d\omega_i} \int_{\Omega^+} L_i(p, \omega_i) f_r(p, \omega_i, \omega_o) \cos \theta_i d\omega_i$$

可以看见,前面一项只考虑遮挡,后面一项是只考虑shading,正好是SM算法所做的工作,所以SM算法本质是用RTR约等式对渲染方程式的近似.

由RTR约等式成立条件可知,以下情况下SM近似是比较准确的结果.

- 当光源是点光源或者directioal lighting时候(small support)是准的.
- shading point的BSDF是光滑的,constant radiance area lighting.(光滑)

顶点着色器

```
void main(void) {
    vFragPos = (uModelMatrix * vec4(aVertexPosition, 1.0)).xyz;
    vNormal = (uModelMatrix * vec4(aNormalPosition, 0.0)).xyz;
    gl_Position = uProjectionMatrix * uViewMatrix * uModelMatrix *
        vec4(aVertexPosition, 1.0);
    vTextureCoord = aTextureCoord;
    vPositionFromLight = uLightMVP * vec4(aVertexPosition, 1.0);
}
```

片段着色器(主体)

```
void main(void) {
    vec3 shadowCoord = vPositionFromLight.xyz / vPositionFromLight.w;
    shadowCoord = (shadowCoord + 1.0) * 0.5;
    float visibility = SM(uShadowMap, vec4(shadowCoord, 1.0));
    //float visibility = PCF(uShadowMap, vec4(shadowCoord, 1.0));
    //float visibility = PCSS(uShadowMap, vec4(shadowCoord, 1.0));
    vec3 phongColor = blinnPhong();
    gl_FragColor = vec4(phongColor * visibility, 1.0);
}
```

片段着色器(shadow map)

```
float SM(sampler2D shadowMap, vec4 shadowCoord){
    float closestDepth = unpack(texture2D(shadowMap, shadowCoord.xy));
    float currentDepth = shadowCoord.z;
    float visibility = currentDepth - 0.001 < closestDepth ? 1.0 : 0.0;
    return visibility;
}
```

3.1.2 Cascade Shadow Mapping

3.2 软阴影

软阴影由面光源产生.

3.2.1 PCSS

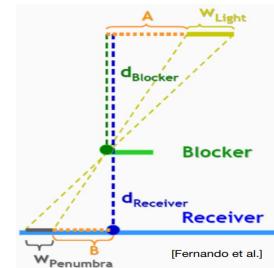
Percentage Closer Filtering(PCF),被引入时是用于anti-aliasing.核心思路是对于世界坐标系上某点 x ,投影到SM上,取SM上周围一圈点的深度值做平均.

PCF的片段着色器

```
float PCF(sampler2D shadowMap, vec4 coords) {
    float bias = 0.002;
    float visibility = 0.0;
    float currentDepth = coords.z;
    float filterSize = 1.0 / 2048.0 * 20.0;
    poissonDiskSamples(coords.xy);
    for(int i=0;i<NUM_SAMPLES;i++){
        vec2 texcoords = poissonDisk[i]*filterSize+coords.xy;
        float closesDepth = unpack(vec4(texture2D(shadowMap,texcoords).xyz,1.0));
        visibility += closesDepth < currentDepth - bias ? 0.0 : 1.0;
    }
    return visibility/float(NUM_SAMPLES);
}
```

不同阴影部分应该要有不同Filter Size产生不同程度软阴影,设 d 表示阴影接受物到阴影投射物的距离(blocker distance,或relative average projected blocker depth),

- d 越小,filter size应该小,阴影硬
- d 越大,filter size应该大,阴影软



$$w_{penumbra} = (d_{Receiver} - d_{Blocker})w_{Light}/d_{Blocker}$$

PCSS算法

1. Blocker Search(在某区域内得到平均blocker depth)
2. Penumbra Estimation(用average blocker depth来获得filter的size)
3. Percentage Closer Filtering(The percentage of exels that are in front of the shading point)

PCSS的片段着色器

```
float findBlocker( sampler2D shadowMap, vec2 uv, float zReceiver ) {
    poissonDiskSamples(uv);
    float depthSum = 0.0;
    int blockSum = 0;
```

```

for(int i = 0; i<NUM_SAMPLES; i++) {
    vec2 uv_ = uv + poissonDisk[i] / 2048.0 * 30.0;
    float shadowMapDepth = unpack(vec4(texture2D(uShadowMap, uv_).rgb, 1.0));
    if(zReceiver > (shadowMapDepth + 0.002)){
        depthSum += shadowMapDepth;
        blockSum++;
    }
}
if(blockSum == 0) return -1.0;
if(blockSum >= NUM_SAMPLES ) return 2.0;
return depthSum / float(blockSum);
}

float PCSS(sampler2D shadowMap, vec4 coords){
    // STEP 1: avgblocker depth
    float blockerDepth = findBlocker(shadowMap, coords.xy, coords.z);
    if(blockerDepth < EPS) return 1.0;
    if(blockerDepth > 1.0+EPS) return 0.0;
    // STEP 2: penumbra size
    float penumbraSize = (coords.z - blockerDepth) / blockerDepth;
    // STEP 3: filtering
    float visibility= 0.0;
    for(int i = 0; i<NUM_SAMPLES ; i++){
        vec2 uv_ = coords.xy + poissonDisk[i] / 2048.0* 30.0 * penumbraSize;
        float shadowMapDepth = unpack(vec4(texture2D(uShadowMap,uv_).rgb,1.0));
        visibility += coords.z > shadowMapDepth + 0.002 ? 0.0 : 1.0;
    }
    return visibility / float(NUM_SAMPLES);
}

```

3.2.2 Variance Soft Shadow Mapping

由于降噪技术提升,最近PCSS渐渐压过VSSM.

PCSS中的第1,3步存在的性能问题,VSSM解决第1,3步,快速计算某区域内平均block depth和Shadow Map一块区域深度的均值和方差.

- 区域内平均值,解决方法:MipMaping,Summed Area Table(SAT)
- 区域内方差: $Var(X) = E(X^2) - E^2(X)$,只要一张额外的平方深度图.

VSSM核心思路: 快速求area内深度的均值和方差,然后利用Chebychev不等式,对于任意分布

$$P(x > t) \leq \frac{\sigma^2}{\sigma^2 + (t - \mu)^2}$$

在RTR中直接认为不等式是近似等于,Chebychev不等式用作约等式, $t > \mu$ 时比较准, $t < \mu$ 时不准但是也直接这样用效果还不错.

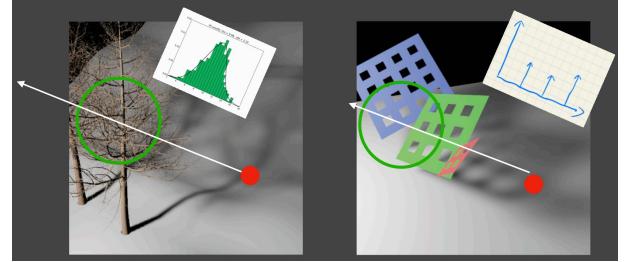
解决PCSS第一步, z_{occ} 代表遮挡物的平均深度,是待求的值, N 表示area像素点数,有 N_1 个是非遮挡物, N_2 个是遮挡物.

$$\frac{N_1}{N} z_{unocc} + \frac{N_2}{N} z_{occ} = z_{Avg}$$

z_{Avg} 通过MipMapping或者SAT可直接得到,由Chebychev得 $\frac{N_1}{N} = P(x > t)$,则 $\frac{N_2}{N} = 1 - \frac{N_1}{N}$,假设非遮挡物的深度都是shading point深度即 $z_{unocc} = t$ (绝对接

收物都是一个平面).则可以求出 z_{occ}, z_{Avg} 可以直接求(像素窗口内求平均)

出现问题的情况(分布不符合正太分布的假设)



3.3 Moment Shadow Mapping

解决VSSM中分布可能不准的情况,引入概率中矩的概念,VSSM本质是利用了前两阶的矩.MSM核心思路是多记录几阶矩. 矩越多,越能准确拟合出PCF,当然内存和计算开销也会越大.效果上是缓解了Light leaking现象.

3.3.1 Distance Field Soft Shadows

SDF shadow 很快,但是存储开销比较大.SDF背后的理论和最优传输有关,得到场景几何的SDF后,从shading point开始向光源做sphere tracing,设shading point是 p ,第 i 个球心是 o_i ,切点是 t_i ,安全角度是 $\theta = \min_i \angle o_i p t_i$,越小的“safe angle”等价于越小的visibility.

$$\angle o_i p t_i = \arcsin \frac{SDF(p)}{p - o_i}$$

由于只需要比较大小,在shader中使用近似公式, k 可以控制阴影软硬.

$$\min \left\{ \frac{k \cdot SDF(p)}{p - o_i}, 1.0 \right\}$$

优点是渲染时快且质量搞,但是需要预算算和很大的存储空间.

4 全局光照

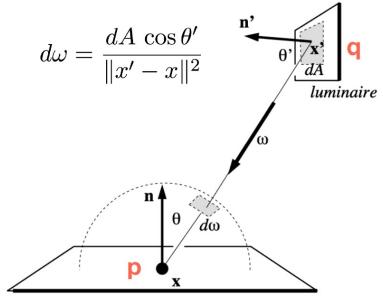
全局光照就是想解决光线多次弹射后的间接光照

4.1 3D Space

4.1.1 Reflective Shadow Map(RSM)算法

首先找出所有次级光源(哪些物体表面的点会被直接光照射到),利用shadow map(每个点光源看作一个camera).

假设次级光源出射辐射度量在各个方向上均匀分布.接下来考虑,已知次级光源,求其对 p 点 L_r 的贡献



$$\begin{aligned} L_r(p, \omega_r) &= \int_{\Omega_{patch}} L_i(p, \omega_i) V(p, \omega_i) f_r(p, \omega_i, \omega_r) \cos \theta_i d\omega_i \\ &= \int_{A_{patch}} L_i(q \rightarrow p) V(p, \omega_i) f_r(p, q \rightarrow p, \omega_r) \frac{\cos \theta_p \cos \theta_q}{\|q - p\|^2} dA \end{aligned}$$

q点是次级光源中心点,在dA面积很小的情况下,可假设积分内函数值为常数.

一个RSM储存每个像素点p的深度值 d_p ,世界坐标系下对应的点坐标 x_p ,法向方向 n_p ,和可见点的反射radiant flux Φ_p ,假设光源无限小且忽视遮挡问题,则从 x_p 向 ω 方向发出的radiant intensity为

$$I_p(\omega) = \Phi_p \max\{0, n_p \cdot \omega\}$$

在surface上的法向为n的点x的irradiance为(注意, $x - x_p$ 没有归一化)

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

$$E(x) = \frac{d\phi(x)}{dA}$$

$$E(x, n) = \sum_{p \in pixels} E_p(x, n)$$

对于场景中每个点需要计算|pixels|次,计算太高昂,故采用采样的方法,原文做了一个大胆的假设,将x投影到屏幕上得 p_x ,认为 p_x 周围一圈范围内的点 $|p - p_x| > Threshold$ 对x间接光照贡献最多

优点是便于实现,缺点是计算复杂度与直接光源的个数呈线性,间接光照没有visibility check.很多假设,如假设diffuse反射物,需要采样与质量之间的权衡.

4.1.2 Light Propagation Volumes(LPV)算法

先使用RSM计算出场景中的虚拟光源(VPL),然后将VPL的光照“注射”给最近的格子,然后再“传播”到整个场景.

对动态场景适用,但是如果格子尺度过大,如果某个遮光板尺度小于格子尺度,则会出现该遮挡而未遮挡情况.

光照注射 对于每一个VPL,将其视作具有special directional intensity distribution $I_p(\omega)$ 的 area light source.

$$I_p(\omega) = \Phi_p \max\{0, n_p \cdot \omega\}$$

使用n bands of SH,共 n^2 个参数 $c_{l,m}$ 来描述基函数 $y_{l,m}(\omega)$.

$$I_p(\omega) \approx \sum_{l,m} c_{l,m} y_{l,m}(\omega)$$

假设某个VPL p所在的LPV cell为i,则

$$I_i(\omega) = I_p(\omega) \approx \sum_{l,m} c_{l,m} y_{l,m}(\omega)$$

几何注射 一般会对多个light source 进行RSM,所以可以得到一个稠密的场景采样,除此之外也可以对场景表面求precomputed point sampling.总之,场景表面3D信息可知.

场景每个采样点可以视作一个有location, orientation and size的surfel(surface element).在网格尺度为s的voxel中,光线被一个面积为 A_s ,法向为 n_s 的surfel遮蔽的概率为

$$V_p(\omega) = A_s s^{-2} \max\{0, n_s \cdot \omega\}$$

同理用球谐表达

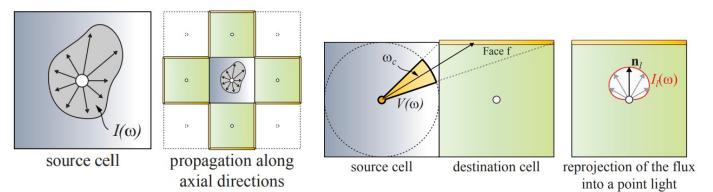
$$V_p(\omega) \approx \sum_{l,m} v_{l,m} y_{l,m}(\omega)$$

光照传播 最开始迭代以LPV injection为输入,后续迭代以上一次LPV为输入,每次迭代中,每个cell只影响轴向6个相邻的cell.

先考虑从cell i辐射能量到相邻cell j的某个face k上

$$\Phi_{i \rightarrow f_{jk}} = \int_{\Omega} I_i(\omega) V_{i \rightarrow f_{jk}}(\omega) d\omega$$

积分可能十分不精确.选用另一种近似策略



将 f_{jk} 向以cell i中心为球心的球投影,得到的立体角为 $\Delta\omega_{i \rightarrow f_{jk}}$

$$\Delta\omega_{i \rightarrow f_{jk}} = \int_{\Omega} V_{i \rightarrow f_{jk}}(\omega) d\omega$$

设 $\omega_{i \rightarrow f_{jk}}$ 是从cell i中心到 f_{jk} 中心的立体角,则到每个面上的辐射强度为

$$\Phi_{i \rightarrow f_{jk}} \approx \frac{1}{4\pi} \Delta\omega_{i \rightarrow f_{jk}} I_i(\omega_{i \rightarrow f_{jk}})$$

cell j 的 k 面上有发出的flux等价于cell j 中心有个虚拟点光源,其法向方向为 $n_{j \rightarrow f_{jk}}$

设 $adj(i)$ 表示cell i 轴向6个相邻cell, $I_j(\omega)$ 表示cell j 内辐射强度分布,则

$$\sum_{i \in adj(j)} \Phi_{i \rightarrow f_{jk}} = \int_{\Omega} \Phi_{j \rightarrow f_{jk}} \max\{0, n_{j \rightarrow f_{jk}} \cdot \omega\} d\omega$$

$$\Phi_{j \rightarrow f_{jk}} = \frac{\sum_{i \in adj(j)} \Phi_{i \rightarrow f_{jk}}}{\pi}$$

$$I_{j \rightarrow f_{jk}}(\omega) = \Phi_{j \rightarrow f_{jk}} \max\{0, n_{j \rightarrow f_{jk}} \cdot \omega\}$$

$$I_j(\omega) = \sum_k I_{j \rightarrow f_{jk}}(\omega)$$

LVP渲染

4.1.3 Voxel Global Illumination(VXGI)算法

场景离散化成格子堆积,建立Hierarchy,第一个pass从light出发,第二次pass对于glossy表面,朝反射方向trace 1 cone ,在Hierarchy上根据cone的大小查询.对于Diffuse surface 多个cone覆盖.

对于动态场景,离散化过程比较慢.

4.2 Screen Space

由在全局光照之前从屏幕能到的信息(直接光照)进行全局光照.

4.2.1 Screen Space Ambient Occlusion(SSAO)

环境光遮蔽(AO)的好处:易于实现,能够大大提升场景中相对位置的感觉.AO是由全局光照造成的.

SSAO的核心思想:

- 不知道间接光照,假设间接光照是常数(同Blinn Phong模型).
- 假设不同点有不同的visibility.
- 假设是diffuse材质.

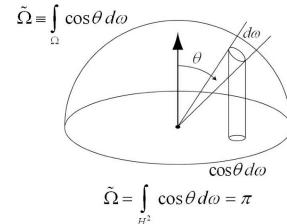
$$L_r(p, \omega_r) = \int_{\Omega} L_i(p, \omega_i) V(p, \omega_i) f_r(p, \omega_i, \omega_r) \cos\theta_i d\omega_i$$

利用RTT approximate拆出Visibility项,间接光 $L_i(\rho)$ 是常数,漫反射BSDF, $f_r = \rho/\pi$.

$$\begin{aligned} L_r(p, \omega_o) &\approx \frac{\int_{\Omega} V(p, \omega_i) \cos\theta_i d\omega_i}{\int_{\Omega} \cos\theta_i d\omega_i} \int_{\Omega} L_i(p, \omega_i) f_r(p, \omega_i, \omega_o) \cos\theta_i d\omega_i \\ &= \frac{\int_{\Omega} V(p, \omega_i) \cos\theta_i d\omega_i}{\pi} \cdot L_i(\rho) \frac{\rho}{\pi} \pi \\ &= k_A(p) L_i(\rho) \frac{\rho}{\pi} \end{aligned}$$

$\cos\theta_i d\omega_i$ 视作一个整体,则对 $d\omega_i$ 在立体角上空间(半球空间)积分等价于 $dx_{\perp} = \cos\theta_i d\omega_i$ 在其投影的半圆空间空间内积分.

Projected Solid Angle

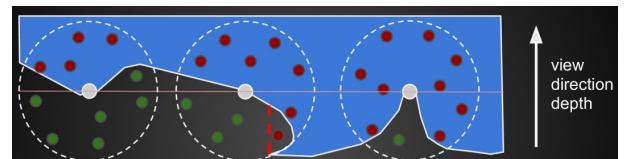


接下讨论如何求解 $k_A(p) = \int_{\Omega} V(p, \omega_i) \cos\theta_i d\omega_i$,AO计算可以在Object Space 和 Screen Space

- Object Space: Raycasting against geometry; slow, require simplifications and spatial data structure;
- Screen Space: Post rendering pass;不需要预处理;与场景复杂度无关;简单,但不是物理上正确的.

对于Screen Space,从Camera看向Scene,认为深度信息是对几何的简单近似,从每个shading point 范围球内开始采样,不知道法向时,假设当红点(深度大于表面深度)个数过半时才开始考虑AO问题.知道法向时(现代都是已知的),可以对半球采样,还可以加权.

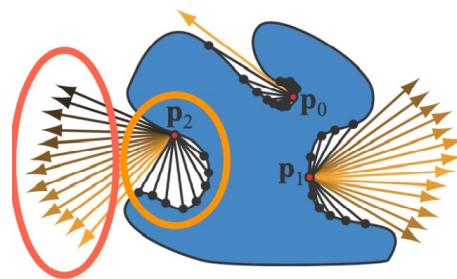
一般只用16个samples,然后denoise.



4.2.2 Horizon Based Ambient Occlusion(HBAO)

与SSAO类似,要求知道着色点的法向量,则只需要在半球区域内采样.

4.2.3 Screen Space Directional Occlusion(SSDO)



- AO是:假设间接光照来自于红色圈中光线,而黄色圈中光线没有间接光照.AO假设环境光来自很远的地方

- DO是:假设红色圈中光线是直接光照,而橙色圈中光线提供间接光照.DO假设环境光来自很近的地方.

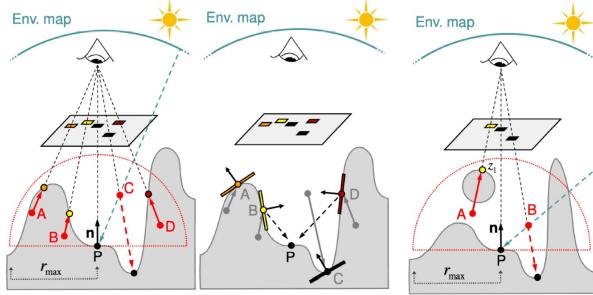
直接光照

$$L_r^{dir}(p, \omega_r) = \int_{\Omega^+, V=1} L_i^{dir}(p, \omega_i) f_r(p, \omega_i, \omega_o) \cos \theta_i d\omega_i$$

间接光照

$$L_r^{indir}(p, \omega_r) = \int_{\Omega^+, V=0} L_i^{indir}(p, \omega_i) f_r(p, \omega_i, \omega_o) \cos \theta_i d\omega_i$$

利用 RSM得到每个像素点的indirect illumination

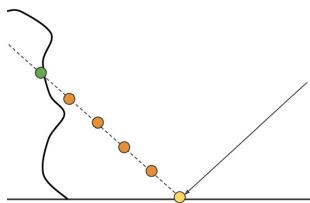


在Shading Point点p中,在p点上半球采样,如图,采样时发现C点没有被遮挡,直接算环境光照A,B,D三点被遮挡,,故这三点在表面上对应位置作为间接光源,对p点着色贡献.只能进行小范围内的GI.

最右边的图,展示了错误的情况,对于错误情况SSDO无法解决,需要用SSR方法解决.

4.2.4 Screen Space Reflection(SSR)

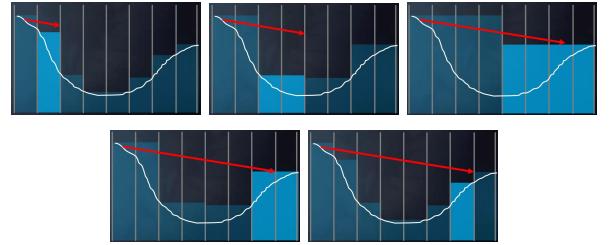
是在屏幕空间上求间接光照,假设漫反射,首先渲染直接光照,对于有反射的表面上的shading point,p,射其法向量为n,相机位置为c,相机能接受到的光线方向为 $\omega_r = c - p$,由n, ω_r 求 ω_i



从p点开始向 ω_i 方向做Linear Raymarch,找到与屏幕空间相对应的点q,用q点作为光源照亮p点.

为了加速,用Hierarchical Tracing方法步长选择,先将场景中的深度做一个mipmap,分别为 Z_1, Z_2, \dots, Z_n ,分辨率逐渐下降, Z_{i-1} 上每个点的像素是 Z_i 上对应四个像素中的最小值.

将深度图看成三维空间上的柱状图,下面系列图展示二维情况下Hierarchical Tracing的过程.



优点是在处理glossy和specular reflection的性能很好,效果好,没有occlusion问题,但在diffuse情况下性能不行,缺少屏幕空间之外的信息

4.3 Radiosity算法

Radiosity算法只可以解漫反射,设 A_i 是element i的面积, B_i 是i的辐照度量, E_i 是射出的辐照强度, ρ_i 是反射率, F_{ji} 是Form Factor.

$$A_i B_i = A_i E_i + \rho_i \sum_{j=1}^n F_{ji} A_j B_j$$

θ, θ' 是x与y连成的向量 r 分别与x处法向量,y处法向量之间的夹角.

$$A_i F_{ij} = \int_{x \in P_i} \int_{y \in P_j} \frac{\cos \theta \cos \theta'}{\pi \|r\|_2^2} v(x, y) dy dx = A_j F_{ji}$$

$$v(x, y) = \begin{cases} 1, & x, y \text{ 之间无遮挡} \\ 0, & \text{otherwise} \end{cases}$$

由Form factor的对称性

$$\begin{aligned} B_i &= E_i + \rho_i \sum_{j=1}^n (F_{ji} A_j / A_i) B_j \\ &= E_i + \rho_i \sum_j F_{ij} B_j \\ B_i - \rho_i \sum_j F_{ij} B_j &= E_i \end{aligned}$$

改写成矩阵型式后可求得 B_i ,然后利用光栅化渲染.

$$\begin{pmatrix} 1 - \rho_1 F_{11} & -\rho_1 F_{12} & \dots & \rho_1 F_{1n} \\ \rho_2 F_{21} & 1 - \rho_2 F_{22} & \dots & \rho_2 F_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ \rho_n F_{n1} & \rho_n F_{n2} & \dots & 1 - \rho_n F_{nn} \end{pmatrix} \begin{pmatrix} B_1 \\ B_2 \\ \vdots \\ B_n \end{pmatrix} = \begin{pmatrix} E_1 \\ E_2 \\ \vdots \\ E_n \end{pmatrix}$$

4.4 光线追踪算法

4.4.1 基本光线追踪算法

用Monte Carlo思路解渲染方程,作为例子设 ω_i 均匀分布,即 $\omega_i \sim Uniform(0, 2\pi)$,则 $p(\omega_i) = 1/2\pi$.

$$L_r(p, \omega_r) = L_e(p, \omega_r) + \frac{1}{N} \sum_{i=1}^N \frac{f_r(p, \omega_i, \omega_r) L_r(p'(p, \omega_i), -\omega_i)(n \cdot \omega_i)}{p(\omega_i)}$$

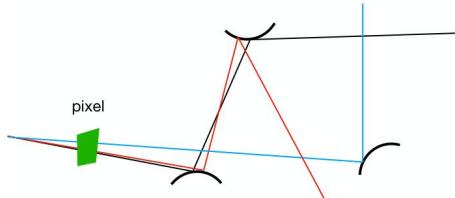
蒙特卡洛光线追踪算法(直接光照).

```
shade(p, wo)
    Randomly choose N directions wi-pdf
    Lo = 0.0
    For each wi
        Trace a ray r(p, wi)
        If ray r hit the light
            Lo += (1 / N) * L_i * f_r * cosine / pdf(wi)
    Return Lo
```

蒙特卡洛光纤追踪算法(全局光照),存在问题,光线数量爆炸,递归调用不会停止.

```
shade(p, wo)
    Randomly choose N directions wi-pdf
    Lo = 0.0
    For each wi
        Trace a ray r(p, wi)
        If ray r hit the light
            Lo += (1 / N) * L_i * f_r * cosine / pdf(wi)
        Else If ray r hit an object at q
            Lo += (1 / N) * shade(q, -wi) * f_r * cosine / pdf(wi)
    Return Lo
```

俄罗斯轮盘赌解决递归调用不停止的问题,即光线以一定概率停止递归.当蒙特卡洛光线追踪,采样光线数目 N 为1时,是路线追踪算法,解决光线数量爆炸的问题,从每个像素,要随机发射出不同方向的path,(当 N 大于1时为分布式追踪).



```
ray_generation(camPos, pixel)
    Uniformly choose N sample positions within the pixel
    pixel_radiance = 0.0
    For each sample in the pixel
        Shoot a ray r(camPos, cam_to_sample)
        If ray r hit the scene at p
            pixel_radiance += 1 / N * shade(p, sample_to_cam)
    Return pixel_radiance

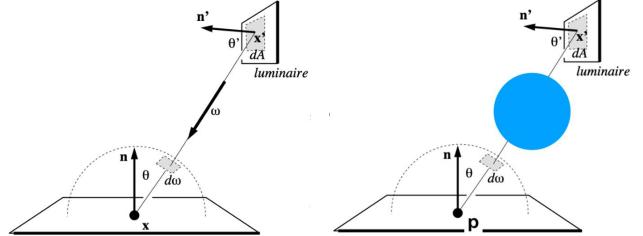
shade(p, wo)
    Manually specify a probability P_RR
    Randomly select ksi in a uniform dist. in [0, 1]
    If (ksi > P_RR) return 0.0;

    Randomly choose ONE direction wi-pdf(w)
    Trace a ray r(p, wi)
    If ray r hit the light
        Return L_i * f_r * cosine / pdf(wi) / P_RR
    Else If ray r hit an object at q
        Return shade(q, -wi) * f_r * cosine / pdf(wi) / P_RR
```

至此,该路径追踪算法是正确且可用的,但是有个低效的问题,因为先前的讨论假设 $\omega_i \sim Uniform(0, \pi)$,如果光源是较小,有大量发出去的光线将被浪费.

解决方案,先将渲染方程改写为在光源上的积分,对于直接光照,直接对光源采样(注意排除遮挡),对于间接光照,按之前做法.

$$d\omega = \frac{dA \cos \theta'}{\|x' - x\|^2}$$



$$L_r(p, \omega_r) = \int_{H^2} f_r(p, \omega_i, \omega_r) L_r(p'(p, \omega_i), -\omega_i)(n \cdot \omega_i) \frac{\cos \theta'}{\|x' - x\|^2} dA$$

```
shade(p, wo)
    # Contribution from the light source.
    Uniformly sample the light at x' (pdf_light = 1 / A)
    L_dir = L_i * f_r * cos theta * cos theta' / |x' - p|^2 / pdf_light

    # Contribution from other reflectors.
    L_indir = 0.0
    Test Russian Roulette with probability P_RR
    Uniformly sample the hemisphere toward wi (pdf_hemi = 1 / 2pi)
    Trace a ray r(p, wi)
    If ray r hit a non-emitting object at q
        L_indir = shade(q, -wi) * f_r * cos theta / pdf_hemi / P_RR

    Return L_dir + L_indir
```

对光源采样时要排除物体遮挡情况

```
# Contribution from the light source.
L_dir = 0.0
Uniformly sample the light at x' (pdf_light = 1 / A)
Shoot a ray from p to x'
If the ray is not blocked in the middle
    L_dir = ...
```

从像素生成ray 一个ray可以由起点 \mathbf{o} 和方向 \mathbf{d} 描述,已知屏幕空间上的点 (x_s, y_s) .

4.4.2 基于降噪技术的实时光线追踪

现代的硬件已经允许我们做1SSP了,1SSP path tracing等于1次rasterization,1次ray(primary visibility),1次ray(secondary bounce),1 ray(secondary vis).但是采用蒙特卡洛积分会得到一个十分noisy的结果(64SSP勉强能看),RTTRT核心技术是降噪,能让1SSP达到十分可观的真实效果.

针对实时渲染不太行的滤波方案

- Sheared Filtering Series(SF,AAF,FSF,MAAF,...)
- 离线滤波技术(IPP,BM3D,APR,...)
- Deep Learning series(CNN,Autoencoder,...), inference速度比较慢,但是之后可能是可行的.

基于时间的滤波 核心思路是当前帧之前的帧是已经滤波好的,要复用,其次使用motion vectors找到前一时刻的location(对于某个物体可以利用motion vector找到前一时刻该物体所在的位置).

G-Buffer(Geometry Buffer),存屏幕空间的信息,如直接光照,Normal,Albedo等.

Back Projection,假设Obj上第k个点,在第i个Frame的屏幕空间上有像素点 p_i ,其对于世界坐标为 x_i^k ,k点第*i*-1个frame上世界坐标为 x_{i-1}^k 投射到屏幕上得到像素 p_{i-1} ,

1. x_i^k 已知(从G-Buffer中获得或者 $x_i^k = M^{-1}V^{-1}P^{-1}E^{-1}x_{i-1}^k$)
2. Motion vector已知,为 $T, x_{i-1}^k = T^{-1}x_i^k$
3. $p_{i-1} = P'V'M'x_{i-1}^k$.
- 4.

$$\bar{C}^{(i)} = \text{Filter}[\tilde{C}^{(i)}]$$

$$5. \alpha = 0.1 \sim 0.2$$

$$\bar{C}^{(i)} = \alpha \bar{C} + (1 - \alpha) C^{(i-1)}$$

这个方法存在的问题.

- 在突然切换场景,镜头或者光照的情况.
- 倒退着走,边缘会逐渐出现新的物体(Screen Space 问题).
- 因为被遮挡物突然变得未遮挡,即场景中突然出现新的物体(产生拖影),调整方案(都会重新引入noise)
 - Clamping:
 - Detection:用Obj ID 检测temporal failure,然后调整 α ,
- shading上的问题,阴影拖影,反射拖影.

基于空间的滤波

联合双边滤波(cross/joint bilateral filter) ,

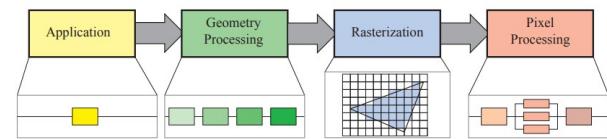
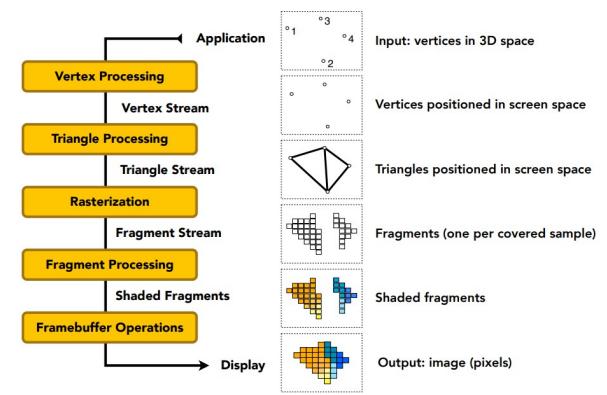
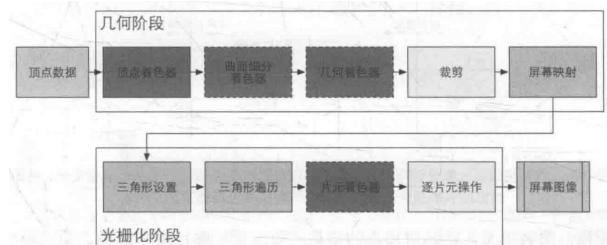
针对实时渲染的特定滤波方法(SVGF)

Recurrent AutoEncoder(RAE)

Part II

实时渲染的实现

5 渲染管线



5.1 Application Stage

一般在CPU上运行,也可以用Computer Shader在GPU上运行.Application Stage结束时将Render Primitives(如,点,线,三角形)传递给Geometry Processing.一般可以做物理仿真,碰撞检测,culling算法等.

5.2 顶点着色器

顶点着色器输入是顶点在模型坐标系下的坐标和其他信息(如法向,uv坐标),输出是顶点在屏幕坐标系下的坐标和相应信息.

rasterization阶段根据顶点在屏幕坐标系下坐标和图元信息生成Fragment,并通过插值得到Fragment对应的颜色,法向,坐标等.

某点坐标 $\mathbf{p}_{obj} = \begin{pmatrix} x_{obj} & y_{obj} & z_{obj} & 1 \end{pmatrix}^T$, 则从模型坐标系到世界坐标系再到clip坐标系可描述为

$$\mathbf{p}_{clip} = \mathbf{M}_{proj} \mathbf{M}_{view} \mathbf{M}_{model} \mathbf{p}_{obj}$$

物体坐标系中法向量转换到摄像机坐标系, 证明见11.5

$$\mathbf{n}_{clip} = ((\mathbf{M}_{proj} \mathbf{M}_{view} \mathbf{M}_{model})^{-1})^T \mathbf{n}_{obj}$$

\mathbf{M}_{view} 推导实际使用时可由glulookAt或glm::lookAt函数获得.

假设有两个坐标系 A, B , 已知 A 的基为 $\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3$, B 的基 $\mathbf{e}^1, \mathbf{e}^2, \mathbf{e}^3$. 假设 B 的基在 A 下的坐标为 $\mathbf{x}_{ab}, \mathbf{y}_{ab}, \mathbf{z}_{ab}$. 则

$$\begin{pmatrix} \mathbf{e}^1 \\ \mathbf{e}^2 \\ \mathbf{e}^3 \end{pmatrix} = (\mathbf{x}_{ab} \quad \mathbf{y}_{ab} \quad \mathbf{z}_{ab}) \begin{pmatrix} \mathbf{e}_1 \\ \mathbf{e}_2 \\ \mathbf{e}_3 \end{pmatrix}$$

$\mathbf{q}_a \in \mathcal{R}^3$ 表示某点在 A 下的坐标, $\mathbf{q}_b \in \mathcal{R}^3$ 表示在 B 下的坐标. 若 A, B 坐标系相对变化可以用旋转矩阵 \mathbf{R}_{ab} 和位移 \mathbf{t}_{ab} 表示.(即, A 旋转 \mathbf{R}_{ab} , 再平移 \mathbf{t}_{ab} 后得到 B)

$$\mathbf{q}_a = \mathbf{R}_{ab} \mathbf{q}_b + \mathbf{t}_{ab}$$

$$\mathbf{q}_b = \mathbf{R}_{ab}^T \mathbf{q}_a - \mathbf{R}_{ab}^T \mathbf{t}_{ab}$$

摄像机坐标系即为 B , 世界坐标系为 A , 给定摄像机位置 \mathbf{c}_p , 摄像机朝向 \mathbf{c}_f , 摄像机上方为 \mathbf{c}_y , 输出

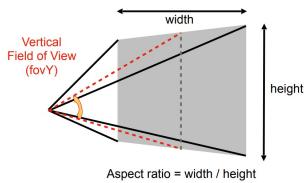
$$\mathbf{M}_{view} = \begin{pmatrix} \mathbf{R}_{ab} & -\mathbf{R}_{ab}^T \mathbf{t}_{ab} \\ \mathbf{0} & 1 \end{pmatrix}$$

$$\mathbf{c}_z = \mathbf{c}_f - \mathbf{c}_p$$

$$\mathbf{R}_{ab} \begin{pmatrix} \mathbf{c}_z \times \mathbf{c}_y & \mathbf{c}_y & -\mathbf{c}_z \end{pmatrix}$$

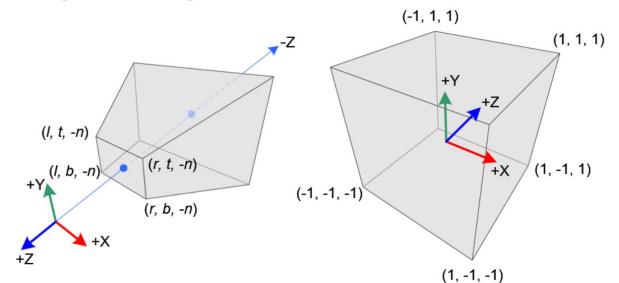
$$\mathbf{t}_{ab} = \mathbf{c}_p$$

\mathbf{M}_{proj} 推导实际使用时可以
用glm::perspective得到 \mathbf{M}_p , 传入参数为vertical fov和
宽高比 aspect



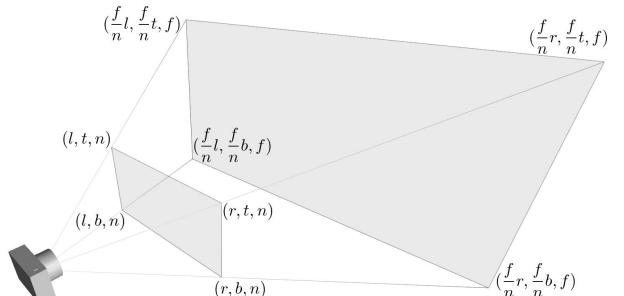
在Viewing Space 定义了一个Frustum区域, 只能看到Frustum区域内的物体. 透视投影矩阵 $\mathbf{M}_{project}$ 的作用是将Frustum区域内坐标映射到右图单位立方体的坐标. 如: $(l, b, -n)$ 映射到 $(-1, -1, -1)$, $(r, t, -n)$ 映射到 $(1, -1, 1)$

Perspective Projection



Perspective Frustum and Normalized Device Coordinates (NDC)

如果给定 l, r, t, b, n, f , 可以写出



$$\mathbf{M}_p = \mathbf{M}_{project} = \begin{pmatrix} \frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & \frac{-(f+n)}{f-n} & \frac{-2f_n}{f-n} \\ 0 & 0 & -1 & 0 \end{pmatrix}$$

性质: 设 $\mathbf{p}_c = \mathbf{M}_p \mathbf{p}_e$, 则有 $w_c = -z_e$, 具体推导见附录.

```
def normalize(arr):
    norm2 = np.linalg.norm(arr)
    return arr/norm2 if norm2 != 0 else arr

# calc view matrix
def lookat(camPos, camFront, camUp):
    camZ = normalize(camFront - camPos)
    camY = normalize(camUp)
    camX = np.cross(camZ, camY)
    M = np.eye(4)
    M[0,0:3] = camX
    M[1,0:3] = camY
    M[2,0:3] = -camZ
    M[0,3] = -camX @ camPos
    M[1,3] = -camY @ camPos
    M[2,3] = camZ @ camPos
    print(M)
    return M

# calc perspective matrix
def perspective(fov, n, f):
    t = 1/tan(fov/2)
    Mp = np.array([[t, 0, 0, 0],
                  [0, t, 0, 0],
                  [0, 0, -(f+n)/(f-n), (-2*f*n)/(f-n)],
                  [0, 0, -1, 0]])
    return Mp

def screen(width, height):
    M = np.eye(4)
    M[0][0]=width/2
    M[1][1]=height/2
    M[0][3]=width/2
    M[1][3]=height/2
    return M
```

```

width = 800
height = 800
camPos = np.array([1.2,2.5,1])
camFront = np.array([0,0,1])
camUp = np.array([0.,0,-1])
Mv = lookat(camPos,camFront,camUp)
Mp = perspective( 2*pi/3 , 0.022 , 0.2*sqrt(2) )
Ms = screen(width, height)
obj = np.array([[0,0,0],[0.2,0,0],[0.2,1.5,0],[0,1.5,0],
[0,0,0.8],[0.2,0,0.8],[0.2,1.5,0.5],
[0,1.5,0.5],[0.2,1,0.5],[0.2,1,0.8],
[0,1,0.5],[0,1,0.8]])
objHomo = np.vstack( (obj.T,np.ones((1,obj.T.shape[1])) ) )
objScreenCorHomo = (Ms @ Mp @ Mv @ objHomo)
objScreenCor = objScreenCorHomo[0:2,:] / objScreenCorHomo[3,:]

```

5.3 可选着色器

- Tessellation shader: 可以控制三角形精细程度, camera距离较远时用较少的三角形渲染, camera较近时用较多的三角形. 由外壳着色器(Hull Shader), 镶嵌器(Tessellator)和域着色器(Domain Shader)构成, 其中外壳和域着色器是可编程的, 镶嵌器硬件管理, 可以借助曲面细分的技术实现细节层次机制(Level of Detail).
- Geometry shader: 集合着色器输入是完整的图元(如:点), 输出可以是一个或多个其他图元(如三角面), 或者不输出图元. 拿手好戏是将输入点或线扩展成多边形. 几何着色器可以用来生成particle, 如渲染火焰时, 将仿真得到的火焰vertex转变为用两个三角面片组成的square, 并将其面向camera.
- Stream Output: 可以在此处输出数据array, 传递给CPU或GPU, 跳过后续渲染操作, 通常用于粒子仿真.

图元组装

Clipping 把顶点数据收集并组装成三角形, 会进行裁剪和背面剔除相关优化, 以减少进入光栅化的图元数量.

在顶点着色器中算出 $\mathbf{p}_c = (x_c \ y_c \ z_c \ w_c)$, 然后硬件在齐次空间进行裁剪(clipping), 即, 保留满足

$$-w_c \leq x_c, \ y_c, \ z_c \leq w_c$$

的点

如果一个三角形部分在空间内, 部分不在, 会在交界处产生新顶点代替原来的旧顶点.

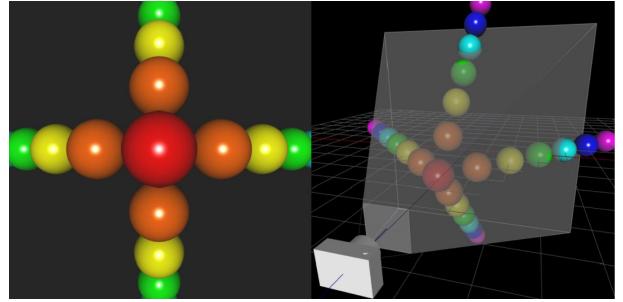
常见的裁剪算法有Cohen-Sutherland算法, Liang-Barsky算法和Sutherland-Hodgman多边形裁剪算法.

Screen Mapping 在光栅化之前, 进行屏幕映射操作: 透视除法和视口变换. 对于保留下来的点, 由硬件进行透视除法得到 $\mathbf{p}_n = (x_n \ y_n \ z_n)$

透视除法和视口变换归于图元组装阶段, 有些资料归于光栅化阶段, 硬件实现所以不同厂商有不同的设计.

为什么先clipping后透视除法?

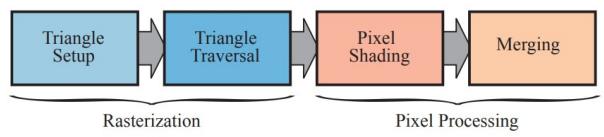
- 效率, 除法开销大
- 齐次裁剪空间下, 节点坐标具有线性关系, 可以进行线性插值运算(裁剪产生新加入顶点替代旧顶点, 新顶点需要通过插值得到顶点上的属性).
- 如果不裁剪, 先进行透视除法, 则可能有三角形中 $z_e = 0$, 此时 $w_c = 0, x_n = x_c/w_c$ 未定义.



在opengl中设置glViewPort让屏幕映射由硬件完成, 假设屏幕宽为 W , 高为 H , 屏幕原点为在左上角, 屏幕坐标系上光心为 (x_c, y_c) (一般为 W, H), 则

$$\mathbf{M}_s = \mathbf{M}_{screen} = \begin{pmatrix} W/2 & 0 & 0 & x_c \\ 0 & H/2 & 0 & y_c \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

5.4 光栅化(Rasterization)



将Primitive进行Screen Map后, 目标是找到所有在Primitive内的pixel, 并渲染.

三角形组装(Triangle Setup) : 将Screen Space上的点连成三角形, 然后进行backface culling, 将没有被culled的传到三角形遍历过程.

三角形遍历(Triangle Traversal) : 对于每个三角形, 创建出被三角形覆盖的片元, 片元数据由顶点数据插值所得(注意, 需要使用Perspective Correct Interpolation, 见附录).

此处注意, 严格来说, 因为对法向进行了插值, 所以法向不再是单位法向, 需要重新标准化, 然而不重新标准化效果差别

不大,所以移动端重视性能就不再重新标准化. 之后对于每个Fragment要进行测试,如果通过测试,传入pixel shader.

5.5 Pixel Processing

Pixel Shading 通过编写Pixel Shader(即OpenGL中的Fragment Shader)完成,每个Fragment执行一次.Fragment is the data necessary to generate a single pixel's worth of a drawing primitive in the frame buffer.

输入是interpolated shading data,输出一个或多个颜色

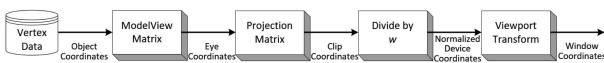
测试混合阶段 又叫逐片元操作(Per-Fragment Operations),逐pixel信息储存在Color Buffer,Merge Stage 需要将Pixel Shading产生的Fragment Color与Color Buffer中的数据结合起来.是一个高度可定制化阶段,有多种效果.

Merge Stage同时解决Visibility问题,Z-buffer与Color Buffer大小一致,渲染时,如果某个pixel的new z-value小于z-value in Z-Buffer,则更新该pixel处color 与 z-value.

一个Frame Buffer包含 color-buffer, depth-buffer, alpha channel(与Color Buffer高度绑定,为每个pixel储存opacity value),stencil Buffer,渲染管线将结果输出到offscreen buffer中,然后实施抗锯齿方法(滤波, 降采样等),将结果输出到Back Buffer, 然后交换到Front Buffer.

半透明物体渲染时一般会使用顺序无关的半透明渲染技术(Order-independent transparency, OIT).

5.6 小结



纯理论模型中,对于世界坐标系中某点 $\mathbf{p} = \begin{pmatrix} x & y & z & 1 \end{pmatrix}^T$,投射到屏幕坐标系中某点 $\mathbf{p}_s = \begin{pmatrix} x_s & y_s & z_s & w_s \end{pmatrix}^T$ 的过程可以描述为

$$\mathbf{M}_s \mathbf{M}_p \mathbf{M}_v \mathbf{M}_m \mathbf{p} = \mathbf{p}_s$$

其中 $w_s = w_c = -z_e$,即该点的深度.

OpenGL应用场景下,假设知道屏幕坐标 (x_s, y_s) ,对应深度 $-z_e$,则

$$\mathbf{p}_s = \left(x_s \quad y_s \quad -\frac{f+n}{f-n} z_e + -\frac{2f}{f-n} \quad -z_e \right)^T$$

$$\mathbf{p} = (\mathbf{M}_s \mathbf{M}_p \mathbf{M}_v \mathbf{M}_m)^{-1} \mathbf{p}_s$$

一个软光栅系统大致思路

```

void rasterize(...){
    for(Triangle& t : triangle_list){
        // mvp transform
        // clipping
        // homogeneous division
        // view port transform
        ...
        rasterize_triangle(t, view_space_pos);
    }
}

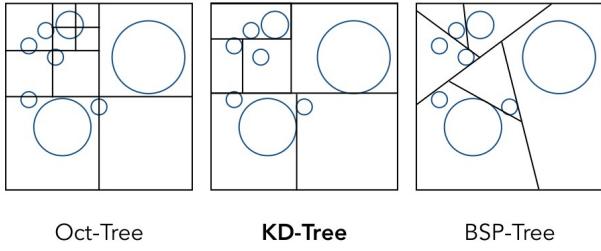
void rasterize_triangle(Triangle& t, array<Vector3f, 3>& view_pos){
    auto v = t.v;
    float minx = FLT_MAX, miny = FLT_MAX, maxx = FLT_MIN, maxy = FLT_MIN;
    // find triangle's AABB
    for (size_t i = 0; i < 3; i++) {
        const Vector4f& p = t.v[i];
        minx = p.x() < minx ? p.x() : minx;
        miny = p.y() < miny ? p.y() : miny;
        maxx = p.x() > maxx ? p.x() : maxx;
        maxy = p.y() > maxy ? p.y() : maxy;
    }

    for(int x = (int)minx;x < maxx;x++) {
        for (int y = (int)miny; y < maxy; y++) {
            if(!insideTriangle(x,y,t.v)) continue;
            auto[a, b, c] = computeBarycentric2D(x, y, t.v);
            // perspective correct interpolation
            float Z = 1.0/(a/v[0].w()+b/v[1].w()+c/v[2].w());
            float zp = a*v[0].z()/v[0].w()+ b*v[1].z()/v[1].w()+c*v[2].z()/v[2].w();
            zp *= Z;
            // z-buffer
            int buf_index = get_index(x,y);
            if(zp >= depth_buf[buf_index]) continue;
            depth_buf[buf_index] = zp;
            // interpolate for color, normal, uv etc
            auto ic=
            interpolate(a,b,c,t.color[0],t.color[1],t.color[2],1);
            auto in=
            interpolate(a,b,c,t.normal[0],t.normal[1],t.normal[2],1);
            auto iuv=
            interpolate(a,b,c,t.tex_coords[0],t.tex_coords[1],t.tex_coords[2],1);
            auto ipos=
            interpolate(a,b,c,view_pos[0],view_pos[1],view_pos[2],1);
            // pixel shading
            fragment_shader_payload payload(ic, in.normalized(), iuv, texture ? &*texture : 0);
            payload.view_pos = ipos;
            auto pixel_color = fragment_shader(payload);
            set_pixel(Vector2i(x,y),pixel_color);
        }
    }
}
  
```

6 常用空间数据结构

6.1 基于空间细分的数据结构

每个节点表达一个空间区域,叶子节点的并集即为整个空间



6.1.1 BSP树

BSP树即Binary space partitioning tree,每次自适应用planes划分整个空间,Primitives会associated with每一个包含它的空间(可能同时被两个节点包含).

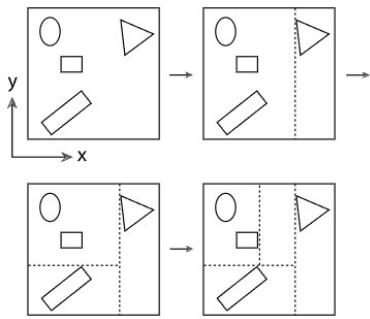
BSP树有很多变种,通常分两类,分别是轴对齐(Axis-Aligned)BSP树和多边形对齐(Polygon-Aligned) BSP树.

KD树和Octree可以视作轴对齐BSP.

6.1.2 KD树

问题,一个物体有可能出现在KD树的多个叶子节点里面,KD树建立并不简单.

KD树的建立



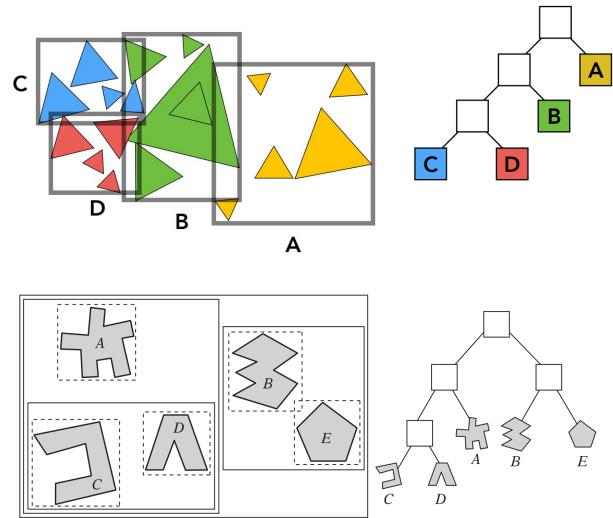
6.1.3 八叉树

6.2 层次包围盒(BVH)

与基于空间细分相对,是基于物体细分的数据结构,Bounding volume(BV)是包含一组物体的volume.BV通常是spheres,AABB(axis-aligned bounding boxes),oriented bounding boxes或者k-DOPs等.



BVH (bounding volume hierarchy)层次包围盒将BV用树的结构来储存,相较于遍历可以降低查找的复杂度.查找方式类似于KD-tree.每次把物体分成两堆.



建立BVH

1. 选择一个dimension分割: 为了让BVH在空间上均匀,可以总是选最长的那根轴分割.
2. 将所有物体按所选dimension排序,将物体均分为两部分.(实现时先求最中间物体,然后以此为参照将每个物体分到不同类)

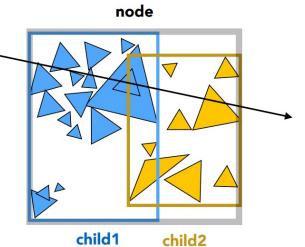
在内部节点上储存Bonding Box,叶子节点上储存Bounding Box和List of Objects.

求光线与BVH中相交的物体

```
Intersect(Ray ray, BVH node) {
    if (ray misses node.bbox) return;
    if (node is a leaf node)
        test intersection with all objs;
        return closest intersection;
    hit1 = Intersect(ray, node.child1);
    hit2 = Intersect(ray, node.child2);
    return the closer of hit1, hit2;
}
```

```
struct BVHBuildNode {
    Bounds bounds;
    BVHBuildNode *left = nullptr;
    BVHBuildNode *right = nullptr;
    Object* obj = nullptr;
};

struct BVHAccel {
    BVHBuildNode* root;
    BVHAccel(vector<Object*> objs) { root = build(objs); }
    BVHBuildNode* build(vector<Object*> objs) {
```



```

BVHBuildNode* node = new BVHBuildNode();
Bounds bounds;
auto l= objs.begin(), m= l + (objs.size() / 2), r= objs.end();
for (int i = 0; i < objs.size(); ++i)
    bounds = bounds.Union(objs[i]->bounds());
if (objs.size() == 1) {
    node->bounds = objs[0]->bounds();
    node->obj = objs[0];
    node->left = nullptr;
    node->right = nullptr;
    return node;
} else if (objs.size() == 2) {
    node->left = build({ objs[0] });
    node->right = build({ objs[1] });
    node->bounds = node->left->bounds.Union(node->right->bounds);
    return node;
} else {
    Bounds cb;
    for (int i = 0; i < objs.size(); ++i)
        cb = cb.Union(objs[i]->bounds().centroid());
    int dim = (cb.p_max - cb.p_min).max_dim();
    if(dim == 0) {
        std::sort(l, r, [](auto f1, auto f2) {
            return f1->bounds().centroid().x < f2->bounds().centroid().x;
        });
    } else if( dim == 1) {
        std::sort(l, r, [](auto f1, auto f2) {
            return f1->bounds().centroid().y < f2->bounds().centroid().y;
        });
    } else {
        std::sort(l, r, [](auto f1, auto f2) {
            return f1->bounds().centroid().z < f2->bounds().centroid().z;
        });
    }
    auto lp= std::vector<Object*>(l, m);
    auto rp= std::vector<Object*>(m, r);
    node->left = build(lp);
    node->right = build(rp);
    node->bounds = node->left->bounds.Union(node->right->bounds);
}
return node;
}

Intersection intersect(BVHBuildNode* node, const Ray& ray) const {
    Intersection isect;
    if (!node->bounds.IntersectP(ray))  return isect;
    if (!node->left && !node->right) return node->obj->intersect(ray);
    Intersection hit1 = intersect(node->left, ray);
    Intersection hit2 = intersect(node->right, ray);
    if (hit1.t && hit2.t){ return hit1.t < hit2.t? hit1 : hit2; }
    else if(hit1.t) return hit1;
    else if(hit2.t) return hit2;
    return isect;
}

void destruct(BVHBuildNode* u){
    if(u == nullptr) return ;
    destruct(u->left);
    destruct(u->right);
    delete u;
}

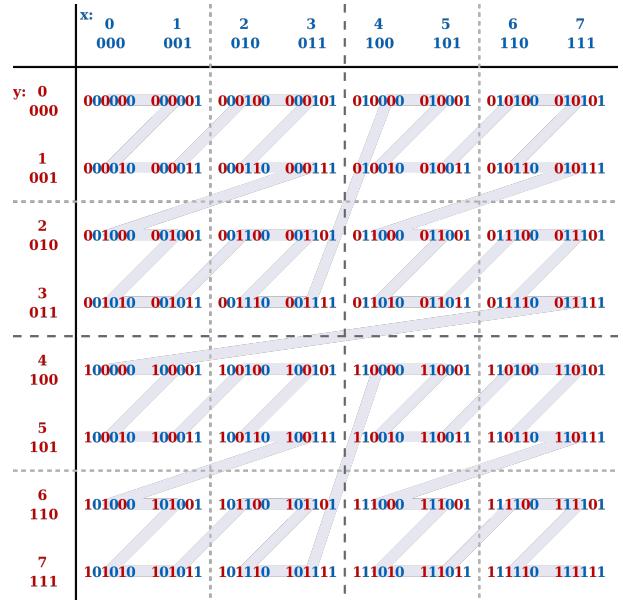
~BVHAccel(){ destruct(root); }
};

```

6.2.1 在GPU上实现BVH

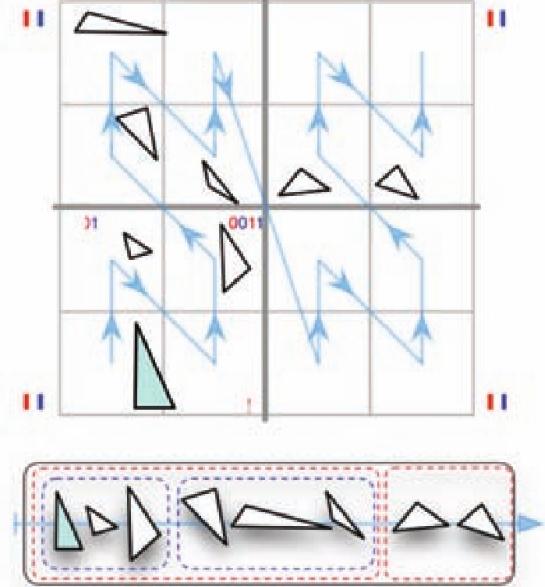
并行建立BVH树

- **Morton Code** 将多维空间数据转换到一维空间,且保留点之间相关次序信息.



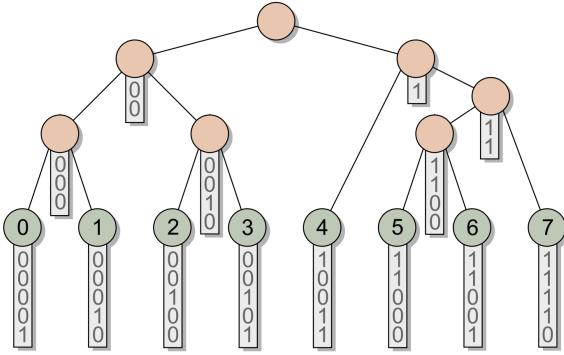
利用用morton code(也叫space-filling Morton curve,z-order curve等)将几何元排序.

- 二叉基数树



建立BVH树的过程

1. 计算每个三角形(几何图元)的AABB,根据所有的AABB求出一个整体AABB(作为根)
2. 对每个三角形对应的AABB,根据其中心点与根AABB的相对位置,求归一化坐标(归一化所有x,y,z坐标到[0,1]区间)



如图所示, a_1, a_2, c 均是三维坐标, c 归一化的坐标可以表达为

$$c_n = \frac{(c - a_2)}{\|a_1 - a_2\|}$$

3. 根据AABB中心点的归一化坐标求三角形对应的Morton Code,以下是3维坐标下Morton Code的求法.

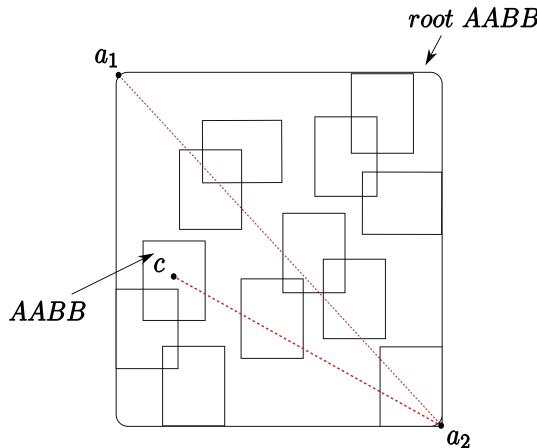
\为某个数的二进制表示中每位后面插入两个0

```
unsigned int expandBits(unsigned int v) {
    v = (v * 0x00010001u) & 0xFF0000FFu;
    v = (v * 0x00000101u) & 0x0F00F00Fu;
    v = (v * 0x00000011u) & 0xC30C30C3u;
    v = (v * 0x00000005u) & 0x49249249u;
    return v;
}
```

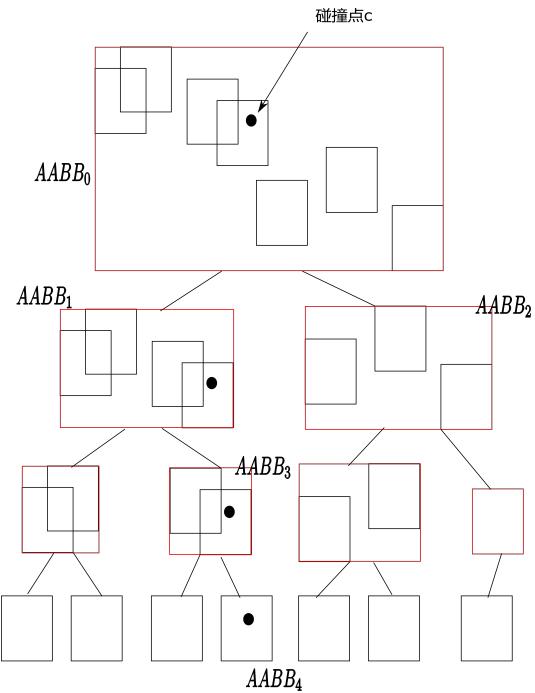
\x,y,z在[0,1]之间,求其Morton Code的算法如下

```
unsigned int morton3D(float x, float y, float z) {
    x = min(max(x * 1024.0f, 0.0f), 1023.0f);
    y = min(max(y * 1024.0f, 0.0f), 1023.0f);
    z = min(max(z * 1024.0f, 0.0f), 1023.0f);
    unsigned int xx = expandBits((unsigned int)x);
    unsigned int yy = expandBits((unsigned int)y);
    unsigned int zz = expandBits((unsigned int)z);
    return xx * 4 + yy * 2 + zz;
}
```

4. 然后根据Morton Code对三角形对应的AABB排序(排序可以采用快速排序或者基数排序,基数排序可并行化).



5. 根据排序好的AABB建立BVH树.建树自底向上,两个叶节点AABB合并生成父节点的AABB,如图,每个节点保存一个AABB信息(叶节点用黑色表示储存的AABB信息,中间节点红色框表示),



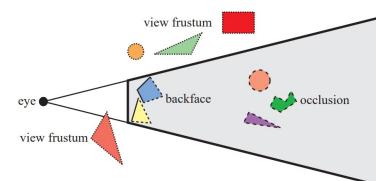
从图中,看查找碰撞时,先检查碰撞点c是否在AABB₀内,若是,则分别判断是否在左右孩子节点的AABB中(AABB₁,AABB₂,如果不在说明不发生碰撞,返回,如果在则继续向下判断,递归直到碰到叶子节点.如果发现在叶子节点的AABB中(AABB₃),则说明很可能碰撞,需要进一步判断.

6. 如果发现碰撞点在AABB₄中,假设AABB₄对应的三角形如下,其代表人体表面的局部,接下来的任务是通过判断点在三角形哪一侧,以判断点是在人体的内部还是外部.

6.3 场景图

7 剔除技术

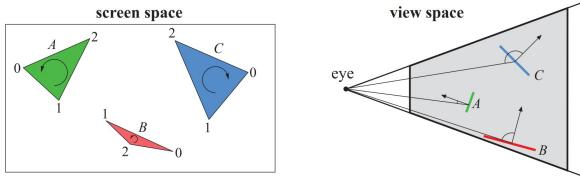
如图所示,Backfacing Culling是剔除法向量背对viewer的面片,View frustum culling剔除不在frustum中的面片,Occlusion Culling剔除因遮挡造成无法看见的面片.



culling在管线中越早发生越好,

Backface Culling 每个面片方向符合右手螺旋法则(大拇指指向法向),如果投影到屏幕空间上面片是顺时针,则该面片是Backface的,所以Backface Culling可以发生在screen-mapping步骤之后.

第二种方法是在三角面片上取一点t,摄像机在v,e = v - t,当e · n < 0时可剔除,其中n是三角面片的法向量.



View Frustum Culling 可以使用Spatial Data Structure,如BVH来加速这个过程.

Detail Culling 当摄像机运动时,放弃一些小的三角形,具体做法是将BV投影到Screen Space,如果投影面积小于某个threshold,则cull.一般在静态时关闭Detail Culling.

Occlusion Culling 一种经典算法是Hierarchical z-buffering(HZB),基本思想是,先对场景建立Octree和Z-buffering frame,对Z-buffering Frame保存 2×2 区域中 $z_{farthest}$ 最大建立Mipmap.

1. 以front-to-back顺序访问octree中节点
2. octree中的Bounding box 投影到Screen Space,再在z-pyramid上使用extended occlusion query进行检测.从coarsest z-pyramid cell开始,将box中的 $z_{nearest}$ 与z-pyramid cell逐点比较,如果所有 $z_{nearest}$ 都大于z-pyramid cell则剔除这个物体.这个过程在z-pyramid上递归直到能够剔除box,或者能够跑遍z-pyramid且不被剔除.
3. 对于可见的Octree box,在Octree上递归最后找到可能需要被渲染的面片,并将这些面片渲染到HZB中,以便能够在接下来的测试中排除更多被遮挡的面片.

现在不会直接使用HZB,大部分算法是

Portal Culling 是一种Occlusion Culling. 场景中的门或窗称作portal,对场景中门或窗使用的view frustum culling机制进行culling.



8 层次细节(LOD)

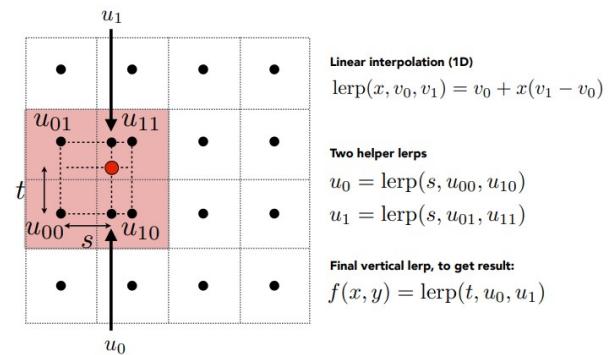
9 纹理

9.1 纹理中的插值算法

如果纹理过小,可以通过Bilinear, Bicubic等插值方法得到较好的效果.

双线性(Bilinear)插值 取周围临近4个,先水平再数值先数值再水平都可以.

Bilinear Interpolation

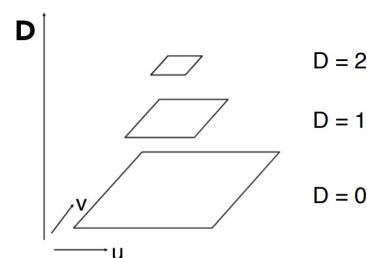


Bicubic插值 取周围临近16个

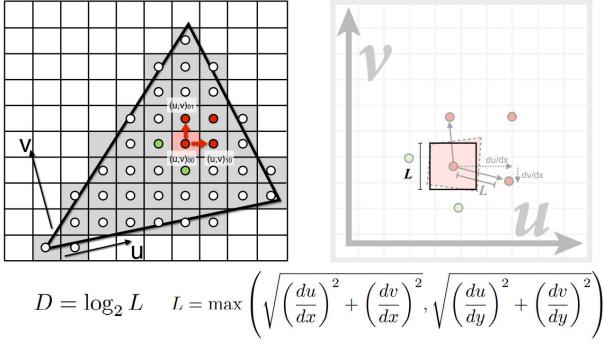
9.2 纹理中的降采样方法

如果纹理过大,对于某些像素点,如果其对应的表面距离相机较远,则这个像素点实际上是对纹理上一块较大的区域采样,根据采样理论,相对来说,此时纹理频率较高,采样频率较低,会出现频谱混叠的问题.这种情况需要对纹理进行降采样,

相机和不同着色点之间的深度不同,所需要降采样的频率也不同,引入Mipmap方法.

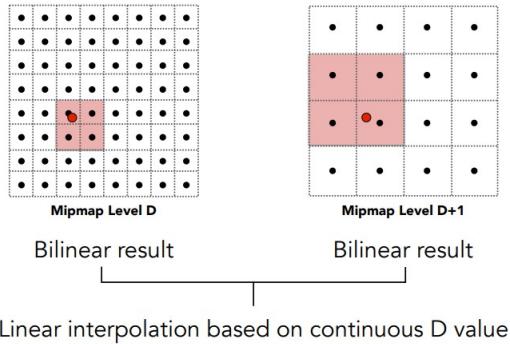


将纹理分成,最底层是原来的纹理,之后往上每层是原来大小的 $1/4$,总存储量为原来存储量的 $4/3$.

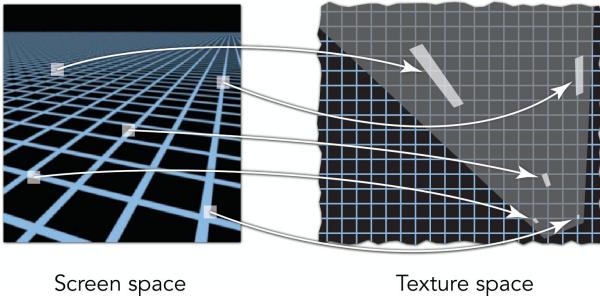


三线性(Trilinear)插值

Trilinear Interpolation



如果像素对映的着色点在纹理上对应的区域不是较为规整的正方形(如斜着的矩形),则此时容易出现过模糊的情况



各项异性过滤可以部分解决这个问题(可以解决矩形区域,但是对于斜着的区域有点效果不太好). EWA filtering 可以解决不规则形状的问题(开销变大).

9.3 Normal Mapping

纹理常见的一种应用是凹凸映射,有两种方式:

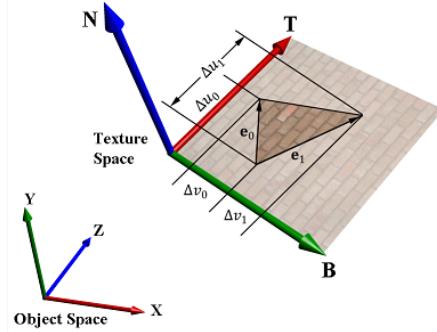
- 高度纹理,利用高度图(一张灰度图)来修改表面的法线,这种方式也称为高度映射(height mapping).
- 法线贴图(normal map), 法线贴图直接存储的就是物体表面的法线

- 物体空间(object space): 物体进行旋转移动操作时,物体空间的法线贴图会得到错误光照,无法复用贴图数据.

- 切线空间(tangent space): z轴总是朝向(0, 0, 1)

高度贴图需要计算物体表面的法线扰动信息,所以说使用法线贴图会比使用高度贴图有更高的性能.

切空间下的法线纹理 纹理坐标(u, v)与位置坐标系可以通过切线空间联系起来,如下,先求得 \mathbf{T} , \mathbf{B} 和 u, v 分别对齐.



$$\nabla U_1 = U_2 - U_1, \quad \nabla V_1 = V_2 - V_1$$

$$\nabla U_2 = U_3 - U_1, \quad \nabla V_2 = V_3 - V_1$$

$$\mathbf{e}_1 = \nabla U_1 \mathbf{T} + \nabla V_1 \mathbf{B}$$

$$\mathbf{e}_2 = \nabla U_2 \mathbf{T} + \nabla V_2 \mathbf{B}$$

$$\begin{pmatrix} \mathbf{e}_{1x} & \mathbf{e}_{1y} & \mathbf{e}_{1z} \\ \mathbf{e}_{2x} & \mathbf{e}_{2y} & \mathbf{e}_{2z} \end{pmatrix} = \begin{pmatrix} \nabla U_1 & \nabla V_1 \\ \nabla U_2 & \nabla V_2 \end{pmatrix} \begin{pmatrix} \mathbf{T}_x & \mathbf{T}_y & \mathbf{T}_z \\ \mathbf{B}_x & \mathbf{B}_y & \mathbf{B}_z \end{pmatrix}$$

$$\begin{pmatrix} \mathbf{T}_x & \mathbf{T}_y & \mathbf{T}_z \\ \mathbf{B}_x & \mathbf{B}_y & \mathbf{B}_z \end{pmatrix} = \begin{pmatrix} \nabla U_1 & \nabla V_1 \\ \nabla U_2 & \nabla V_2 \end{pmatrix}^{-1} \begin{pmatrix} \mathbf{e}_{1x} & \mathbf{e}_{1y} & \mathbf{e}_{1z} \\ \mathbf{e}_{2x} & \mathbf{e}_{2y} & \mathbf{e}_{2z} \end{pmatrix}$$

$$= \frac{\begin{pmatrix} \nabla V_2 & -\nabla V_1 \\ -\nabla U_2 & \nabla U_1 \end{pmatrix}}{\nabla U_1 \nabla V_2 - \nabla U_2 \nabla V_1} \begin{pmatrix} \mathbf{e}_{1x} & \mathbf{e}_{1y} & \mathbf{e}_{1z} \\ \mathbf{e}_{2x} & \mathbf{e}_{2y} & \mathbf{e}_{2z} \end{pmatrix}$$

为每个顶点计算 \mathbf{T} , \mathbf{B} , \mathbf{N} ,给定一张normal mapping纹理 \mathbf{n}'_p (定义在切线空间下),输出着色点 p 法向量为 \mathbf{n}_p .

$$\mathbf{n}_p = (\mathbf{T} \quad \mathbf{B} \quad \mathbf{N}) \mathbf{n}'_p$$

高度纹理 假设高度图为 H, a 经验上取0.01

$$d\mathbf{U}_{ij} = (1, 0, aH_{i+1,j} - aH_{i-1,j})$$

$$d\mathbf{V}_{ij} = (0, 1, aH_{i,j+1} - aH_{i,j-1})$$

所以可以通过叉乘得到点 i, j 处的法向量

$$\mathbf{n} = \frac{d\mathbf{U} \times d\mathbf{V}}{\|d\mathbf{U} \times d\mathbf{V}\|} = \frac{(-d\mathbf{U}_z, -d\mathbf{V}_z, 1)}{\sqrt{d\mathbf{U}_z^2 + d\mathbf{V}_z^2 + 1}}$$

10 实际渲染管线中的技术

10.1 高效Shading算法

Deffered Shading 在计算光照之前做所有的可见性测试，包含两个阶段

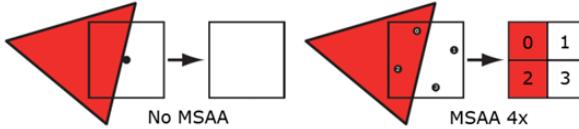
1. 几何处理阶段：先渲染场景一次，之后获得对象的各种几何信息(如，位置，法向量，颜色等)存在G-buffer纹理中。
2. 光照处理阶段：使用G-Buffer中几何数据对每一个片段计算场景光照。

这种方法带来的缺陷是提高了MSAA的成本，在defered shading中使用MSAA,第一阶段生成的G-Buffers是multisampled 数据(不能resolve)，则lightin pass需要读入并处理multisample operation, 如4xMSAA，则fragment shader每个pixel需要运行8次。

10.2 抗锯齿算法

超级采样抗锯齿(SSAA) 超级采样抗锯齿(Super-Sampling Anti-Aliasing, 简称 SSAA),简单效果好,但是计算量巨大,例如: $4 \times SSAA$, 假设屏幕分辨率为 800×600 ,那么需要将屏幕渲染到 1600×1200 的缓冲区上,然后在下采样到 800×600 ,SSAA可以得到非常好的抗锯齿效果。

多重采样抗锯齿(MSAA)



一般光栅化阶段,每个片段都有一个采样点,MSAA中使用多个采样点(记录遮挡与不遮挡),上图 $4 \times MSAA$ 三角形覆盖了4个采样点的两个,最终得到的颜色值要乘以点覆盖率0.5. MSAA 对资源的消耗需求大大减弱(4xMSAAA空间消耗与4xSSAA一样,都是原图四倍,但是只用片段着色器渲染一次,而不是4次),不过在画质上可能稍有不如 SSAA.

快速近似抗锯齿FXAA Fast Approximate AA, 识别出边界,将边界替换成抗锯齿后的边界

TAA Temporal AA,利用上一帧的信息

其他抗锯齿方法 覆盖采样抗锯齿(CSAA) 高分辨率抗锯齿(HRAA) , 可编程过滤抗锯齿(CFAA), 形态抗锯齿(MLAA), 快速近似抗锯齿(FXAA), 时间性抗锯齿(TXAA), 多帧采样抗锯齿(MFAA).

10.3 图像后处理

高斯滤波

$$G(x, y) = G(x)G(y)$$

$$G(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{x^2}{2\sigma^2}}$$

$$\begin{aligned} C_{lm} &= \sum_{i=-1}^4 \sum_{j=-4}^4 G(i, j)C_{l+i, m+j} \\ &= \sum_{i=-1}^4 G(i) \sum_{j=-4}^4 G(j)C_{l+i, m+j} \end{aligned}$$

```
// OpenGL部分
char uniName[20];
float weights[5], sum, sigma2 = 4.0f;
// Compute and sum the weights
weights[0] = gauss(0,sigma2); // The 1-D Gaussian
// function
sum = weights[0];
for( int i = 1; i < 5; i++ ) {
weights[i] = gauss(i, sigma2);
sum += 2 * weights[i];
}

// Normalize the weights and set the uniform
for( int i = 0; i < 5; i++ ) {
snprintf(uniName, 20, "Weight[%d]", i);
prog.setUniform(uniName, weights[i] / sum);
}
```

```
// Fragment Shader
in vec2 TexCoord; // Texture coordinate
uniform sampler2D Texture0;
uniform int Width; // Width of the screen in pixels
uniform int Height; // Height of the screen in pixels
vec4 pass1(){ // 结果存在一个纹理中
    float dy = 1.0 / float(Height);
    vec4 sum = texture(Texture0, TexCoord) * Weight[0];
    for( int i = 1; i < 5; i++ ) {
        sum += texture( Texture0, TexCoord + vec2(0.0,
            PixOffset[i]) * dy )
            * Weight[i];
        sum += texture( Texture0, TexCoord - vec2(0.0,
            PixOffset[i]) * dy )
            * Weight[i];
    }
    return sum;
}
vec4 pass3(){ // 结果存在一个纹理中
    float dx = 1.0 / float(Width);
    vec4 sum = texture(Texture0, TexCoord) * Weight[0];
    for( int i = 1; i < 5; i++ ) {
        sum += texture( Texture0,
            TexCoord + vec2(PixOffset[i], 0.0) * dx )
            * Weight[i];
        sum += texture( Texture0,
            TexCoord - vec2(PixOffset[i], 0.0) * dx )
            * Weight[i];
    }
}
```

```

    }
    return sum;
}
void main(){
    FragColor = pass()
}

```

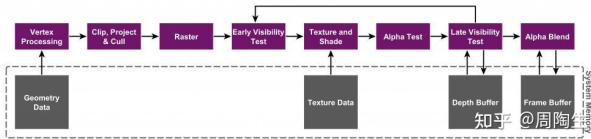
10.4 Tone Mapping

采用HDR渲染出来的亮度值会超过显示器能够显示的最大值, 此时需要将光照结果从HDR转换为显示器能够显示的LDR, 这一过程称为Tone Mapping.

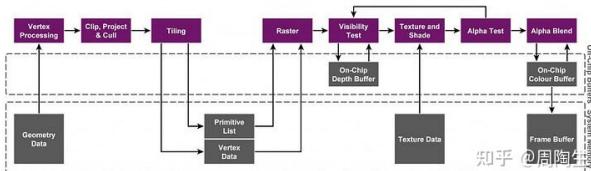
Reinhard tone mapping 如果使用线性映射, HDR到LDR过程中会使亮的地方过亮, 使暗的地方过暗. Reinhard tone mapping 是个经验公式.

10.5 渲染架构

IMR(Immediate Mode Rendering) 每一次渲染API的调用都会直接绘制图形对象, 每次物体(每个Mesh)颜色和深度渲染, 都要读写Frame Buffer 和 Depth Buffer. IMR架构需要大量的带宽, 这点可以用L1, L2缓存优化, 但是移动端的GPU不能将Cache做得太大, 因此移动端一般不可能用IMR.



TBR(Tile-Based Rendering) 移动设备显卡多使用TBR, 其核心思想是将帧缓存分割成一小块一小块, 然后逐块进行渲染.

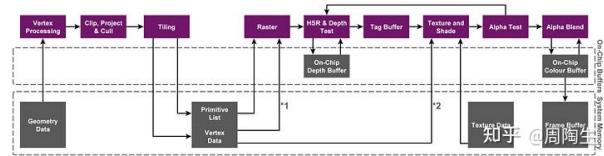


1. 将Primitive List理解为一个哈希表, key是Tile, value是该tile中含有的所有三角形, 在Tiling Pass 中根据每个三角形在Frame Buffer中的位置, 构建Primitive List.
2. 以Tile 为单位执行(等所有三角形完成第一阶段), 从Primitive List取出当前渲染Tile的三角形列表, 对三角形光栅化.
3. 像素着色, 可以Deferred Rendering一个Tile一个Tile地处理其中像素.

由于一个三角形可能在多个Tile中, 所以TBR存在overdraw地问题.

TBDR(Tile Based Deferred Rendering)

TBDR在TBR的基础上, 通过硬件层面的特性HSR解决了Overdraw问题.



相比于TBR, TBDR多了个HSR(隐藏面消除阶段)和TagBuffer在硬件上做到0Overdraw, 基本思想是, 不立即绘制通过Early Z Test 的像素, 而只标记. HSR具体过程

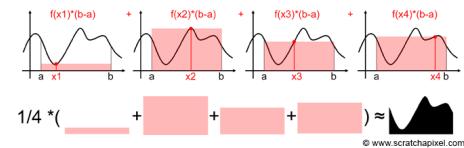
1. 每个fragment, 通过Early Z test后, 先不绘制, 而是HSR阶段读primitive list, 标记该像素由哪个图元绘制.
2. Fragment Shader 绘制时, 只绘制标记这个像素点, 且最终通过Early Z Test 的那个fragment.

Part III

数学

11 RTR中的一些基础知识与推导

11.1 Mente Carlo积分



$$F = \int_a^b f(x) dx$$

$$F = \frac{1}{N} \sum_{i=1}^N \frac{f(x_i)}{p(x_i)}$$

11.2 Rodrigues' Rotation Formula推导

解决的问题: 3维空间中,给定旋转轴 ω ,和旋转角度 θ , 求满足同样旋转的旋转矩阵 R .

设旋转体上有点 q ,此时若以 ω 为单位旋转向量,则线速度

$$\frac{\partial q(t)}{\partial t} = \omega \times q(t) = \hat{\omega} q(t)$$

则微分方程解为

$$\mathbf{q}(t) = e^{\hat{\omega}t} \mathbf{q}(0)$$

如果以 ω 为轴以单位角速度旋转 θ 时间,则

$$\hat{\boldsymbol{\omega}} = \begin{pmatrix} 0 & -a_3 & a_2 \\ a_3 & 0 & -a_1 \\ -a_2 & a_1 & 0 \end{pmatrix}$$

$$\hat{\boldsymbol{\omega}}^2 = \boldsymbol{\omega}\boldsymbol{\omega}^T - \|\boldsymbol{\omega}\|^2 \mathbf{I}$$

$$\hat{\boldsymbol{\omega}}^3 = -\|\boldsymbol{\omega}\|^2 \hat{\boldsymbol{\omega}}$$

$$\begin{aligned} \mathbf{R}(\boldsymbol{\omega}, \theta) = e^{\hat{\boldsymbol{\omega}}\theta} &= \mathbf{I} + \theta \hat{\boldsymbol{\omega}} + \frac{\theta^2}{2!} \hat{\boldsymbol{\omega}}^2 + \frac{\theta^3}{3!} \hat{\boldsymbol{\omega}}^3 \dots \\ &= \mathbf{I} + (\theta - \frac{\theta^3}{3!} + \frac{\theta^5}{5!} + \dots) \hat{\boldsymbol{\omega}} + (\frac{\theta^2}{2!} - \frac{\theta^4}{4!} + \frac{\theta^6}{6!} + \dots) \boldsymbol{\omega}^2 \\ &= \mathbf{I} + \hat{\boldsymbol{\omega}} \sin \theta + \hat{\boldsymbol{\omega}}^2 (1 - \cos \theta) \end{aligned}$$

11.3 重心坐标

三角形上三个顶点坐标为 $\mathbf{a}, \mathbf{b}, \mathbf{c}$,三角形所在平面任意一点为 \mathbf{p} ,则

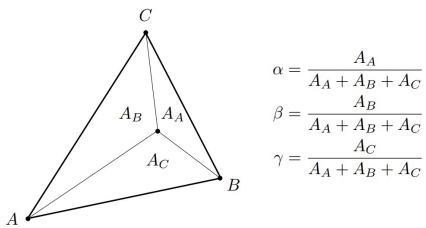
$$\mathbf{p} = \alpha \mathbf{a} + \beta \mathbf{b} + \gamma \mathbf{c}$$

$$\alpha + \beta + \gamma = 1$$

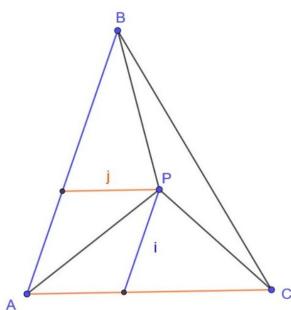
证明:

$$\begin{aligned} \mathbf{p} &= \mathbf{a} + \beta(\mathbf{b} - \mathbf{a}) + \gamma(\mathbf{c} - \mathbf{a}) \\ &= (1 - \beta - \gamma)\mathbf{a} + \beta\mathbf{b} + \gamma\mathbf{c} \end{aligned}$$

若 α, β, γ 均大于0,则 \mathbf{p} 在三角形内部,若其中一个为0,则 \mathbf{p} 在三角形的边或者边的延长线上,若其中两个为0.



证明:假设 \mathbf{j} 与 \mathbf{AC} 平行, \mathbf{i} 与 \mathbf{AB} 平行



$$S_{\triangle ABC} = \frac{1}{2} \|\vec{AB}\| \|\vec{AC}\| \sin \angle \vec{AB}, \vec{AC} \,$$

$$S_{\triangle ABP} = \frac{1}{2} \|\vec{AB}\| \|\mathbf{j}\| \sin \angle \vec{AB}, \mathbf{j} = \frac{1}{2} \gamma \|\vec{AB}\| \|\vec{AC}\| \sin \angle \vec{AB}, \vec{AC} \,$$

$$S_{\triangle APC} = \frac{1}{2} \|\mathbf{i}\| \|\vec{AC}\| \sin \angle \mathbf{i}, \vec{AC} = \frac{1}{2} \beta \|\vec{AB}\| \|\vec{AC}\| \sin \angle \vec{AB}, \vec{AC} \,$$

因此

$$\frac{S_{\triangle APC}}{S_{\triangle ABC}} = \beta, \quad \frac{S_{\triangle ABP}}{S_{\triangle ABC}} = \gamma.$$

三角形的重心满足 $\alpha = \beta = \gamma = \frac{1}{3}$,可以将三角形分成三个面积相等的小三角形

从笛卡尔坐标系转换到重心坐标系,给定三角形三个顶点坐标 $\mathbf{r}_1, \mathbf{r}_2, \mathbf{r}_3$,其中 $\mathbf{r}_i = (x_i, y_i)$,给定三角形平面中某点坐标 \mathbf{r} .求 $\lambda_1, \lambda_2, \lambda_3$ 令 $\mathbf{r} = \lambda_1 \mathbf{r}_1 + \lambda_2 \mathbf{r}_2 + \lambda_3 \mathbf{r}_3$

$$x = \lambda_1 x_1 + \lambda_2 x_2 + \lambda_3 x_3 = \lambda_1 x_1 + \lambda_2 x_2 + (1 - \lambda_1 - \lambda_2) x_3$$

$$y = \lambda_1 y_1 + \lambda_2 y_2 + \lambda_3 y_3 = \lambda_1 y_1 + \lambda_2 y_2 + (1 - \lambda_1 - \lambda_2) y_3$$

整理后

$$\lambda_1(x_1 - x_3) + \lambda_2(x_2 - x_3) + x_3 - x = 0$$

$$\lambda_1(y_1 - y_3) + \lambda_2(y_2 - y_3) + y_3 - y = 0$$

即

$$\mathbf{T} \begin{pmatrix} \lambda_1 \\ \lambda_2 \end{pmatrix} = \mathbf{r} - \mathbf{r}_3$$

$$\mathbf{T} = \begin{pmatrix} x_1 - x_3 & x_2 - x_3 \\ y_1 - y_3 & y_2 - y_3 \end{pmatrix}$$

解可表达为

$$\begin{pmatrix} \lambda_1 \\ \lambda_2 \end{pmatrix} = \mathbf{T}^{-1} \begin{pmatrix} x_1 - x_3 & x_2 - x_3 \\ y_1 - y_3 & y_2 - y_3 \end{pmatrix}$$

$$\lambda_3 = 1 - \lambda_1 - \lambda_2$$

也可以写成显式解

$$\begin{aligned} \lambda_1 &= \frac{(y_2 - y_3)(x - x_3) + (x_3 - x_2)(y - y_3)}{\det(\mathbf{T})} \\ &= \frac{(y_2 - y_3)(x - x_3) + (x_3 - x_2)(y - y_3)}{(y_2 - y_3)(x_1 - x_3) + (x_3 - x_2)(y_1 - y_3)} \end{aligned}$$

$$\lambda_1 = \frac{(y_3 - y_1)(x - x_3) + (x_1 - x_3)(y - y_3)}{\det(\mathbf{T})}$$

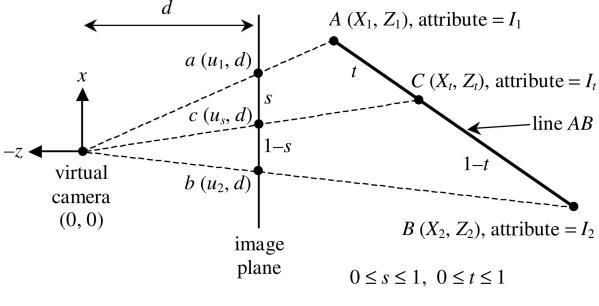
$$= \frac{(y_3 - y_1)(x - x_3) + (x_1 - x_3)(y - y_3)}{(y_2 - y_3)(x_1 - x_3) + (x_3 - x_2)(y_1 - y_3)}$$

$$\lambda_3 = 1 - \lambda_1 - \lambda_2$$

将三角形投影到某个平面后 \mathbf{p}' 对应重心坐标 $(\alpha', \beta', \gamma')$ 与原重心坐标不一样.所以,三维信息(如:深度)不能用投影后的重心坐标插值,而应该用3为空间的重心坐标做插值.

11.4 Perspective Correct Interpolation

先以线段为例,camera空间中的线段坐标为三点为 A, B, C 坐标,通过投影变换为2D空间上三点 a, b, c ,



$$u_s = u_1 + s(u_2 - u_1)$$

$$X_t = X_1 + t(X_2 - X_1)$$

$$Z_t = Z_1 + t(Z_2 - Z_1)$$

如果 O, c, C 三点共线,则

$$t = \frac{sZ_1}{sZ_1 + (1-s)Z_2}$$

对于三角形 \triangle 三点为 $\mathbf{A}, \mathbf{B}, \mathbf{C}$,其中一点为 \mathbf{P} ,对应的barycentric weights是 α, β, γ ,假设通过仿射变换后得到三角形 \triangle' 其三点为 $\mathbf{A}', \mathbf{B}', \mathbf{C}'$,仿射变换矩阵为 \mathbf{M} , \mathbf{P} 点对应 \mathbf{P}' ,在 \triangle' 中的barycentric weights是 α', β', γ' .

$$\begin{pmatrix} \mathbf{A}'w_a \\ w_a \end{pmatrix} = \mathbf{M} \begin{pmatrix} \mathbf{A} \\ 1 \end{pmatrix} \quad \begin{pmatrix} \mathbf{B}'w_b \\ w_b \end{pmatrix} = \mathbf{M} \begin{pmatrix} \mathbf{B} \\ 1 \end{pmatrix}$$

$$\begin{pmatrix} \mathbf{C}'w_c \\ w_c \end{pmatrix} = \mathbf{M} \begin{pmatrix} \mathbf{C} \\ 1 \end{pmatrix} \quad \begin{pmatrix} \mathbf{P}'w_p \\ w_p \end{pmatrix} = \mathbf{M} \begin{pmatrix} \mathbf{P} \\ 1 \end{pmatrix}$$

$$\mathbf{P} = \alpha\mathbf{A} + \beta\mathbf{B} + \gamma\mathbf{C}$$

$$\mathbf{P}' = \alpha'\mathbf{A}' + \beta'\mathbf{B}' + \gamma'\mathbf{C}'$$

已知 α, β, γ ,求 α', β', γ'

$$\mathbf{M} \begin{pmatrix} \mathbf{P} \\ 1 \end{pmatrix} = \alpha\mathbf{M} \begin{pmatrix} \mathbf{A} \\ 1 \end{pmatrix} + \beta\mathbf{M} \begin{pmatrix} \mathbf{B} \\ 1 \end{pmatrix} + \gamma\mathbf{M} \begin{pmatrix} \mathbf{C} \\ 1 \end{pmatrix}$$

$$\begin{pmatrix} \mathbf{P}'w_p \\ w_p \end{pmatrix} = \begin{pmatrix} \alpha\mathbf{A}'w_a + \beta\mathbf{B}'w_b + \gamma\mathbf{C}'w_c \\ \alpha w_a + \beta w_b + \gamma w_c \end{pmatrix}$$

$$\begin{aligned} \mathbf{P}' &= \frac{\alpha w_a}{\alpha w_a + \beta w_b + \gamma w_c} \mathbf{A}' + \frac{\beta w_b}{\alpha w_a + \beta w_b + \gamma w_c} \mathbf{B}' \\ &\quad + \frac{\gamma w_c}{\alpha w_a + \beta w_b + \gamma w_c} \mathbf{C}' \\ \alpha' &= \frac{\alpha w_a}{\alpha w_a + \beta w_b + \gamma w_c} \end{aligned}$$

$$\beta' = \frac{\beta w_b}{\alpha w_a + \beta w_b + \gamma w_c}$$

$$\gamma' = \frac{\gamma w_c}{\alpha w_a + \beta w_b + \gamma w_c}$$

已知 α', β', γ' ,求 α, β, γ .

$$k = \frac{1}{\alpha w_a + \beta w_b + \gamma w_c}$$

$$\alpha' = \alpha w_a k, \quad \beta' = \beta w_b k, \quad \gamma' = \gamma w_c k$$

$$\alpha = \frac{\alpha'}{w_a k}, \quad \beta = \frac{\beta'}{w_b k}, \quad \gamma = \frac{\gamma'}{w_c k}$$

$$1 = \alpha + \beta + \gamma = \frac{\alpha'}{w_a k} + \frac{\beta'}{w_b k} + \frac{\gamma'}{w_c k}$$

$$k = \frac{\alpha'}{w_a} + \frac{\beta'}{w_b} + \frac{\gamma'}{w_c}$$

$$\alpha = \frac{\frac{\alpha'}{w_a}}{\frac{\alpha'}{w_a} + \frac{\beta'}{w_b} + \frac{\gamma'}{w_c}}$$

$$\beta = \frac{\frac{\beta'}{w_b}}{\frac{\alpha'}{w_a} + \frac{\beta'}{w_b} + \frac{\gamma'}{w_c}}$$

$$\gamma = \frac{\frac{\gamma'}{w_c}}{\frac{\alpha'}{w_a} + \frac{\beta'}{w_b} + \frac{\gamma'}{w_c}}$$

11.5 Surface Normal Transform

假设某法向量为 (a, b, c) 的平面方程为

$$ax + by + cz + d = \begin{pmatrix} a & b & c & d \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \mathbf{n}^T \mathbf{p} = 0$$

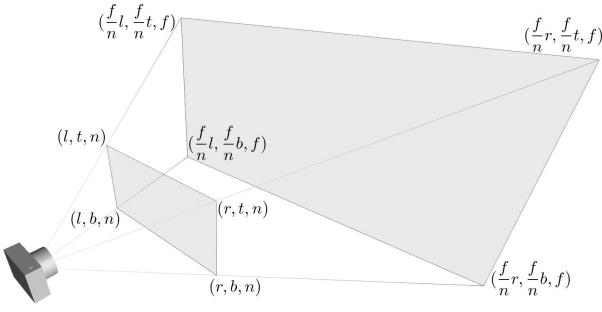
将平面按照 $\mathbf{M} \in \mathcal{R}^{4 \times 4}$ 变换,则 $\mathbf{p}' = \mathbf{M}\mathbf{p}$,假设 \mathbf{n} 对应需按照 $\mathbf{M}' \in \mathcal{R}^{4 \times 4}$ 变换,则

$$\mathbf{n}^T \mathbf{p} = \mathbf{n}^T \mathbf{p}' = (\mathbf{M}' \mathbf{n})^T \mathbf{M} \mathbf{p} = \mathbf{n}^T \mathbf{M}'^T \mathbf{M} \mathbf{p} = 0$$

$$\mathbf{M}' = (\mathbf{M}^{-1})^T$$

有的时候 $\mathbf{M}^{-1} = adj(\mathbf{M})/det(\mathbf{M})$ 当 $det(\mathbf{M}) = 0$ 时没有解,而 $adj(\mathbf{M})$ 与 \mathbf{M}^{-1} 对向量作用区别在于模长,故用 $adj(\mathbf{M})$ 作为替代,最后对法向量单位化即可.

11.6 Perspective Projection推导



对于某个点 $(x, y, z, 1)$,通过与model矩阵,camera矩阵相乘后变换到view空间,其坐标为 $(x_e, y_e, z_e, 1)$.

后与projection 矩阵 \mathbf{P} 相乘(本节所求)后输出 (x_c, y_c, z_c, w_c) (clip coordinate),clipping操作后最终输出点 (x_n, y_n, z_n) (normalized device coordinate,NDC),裁剪掉 $(-1, -1, -1) \sim (1, 1, 1)$ 立方体之外的点.

$$(x_n, y_n, z_n) = (x_c/w_c, y_c/w_c, z_c/w_c)$$

$$\begin{pmatrix} x_c \\ y_c \\ z_c \\ w_c \end{pmatrix} = \mathbf{P} \begin{pmatrix} x_e \\ y_e \\ z_e \\ 1 \end{pmatrix}$$

将 $(x_e, y_e, z_e, 1)$ 其投射到近平面上点 $(x_p, y_p, -n)$ 上.易得

$$x_p = \frac{nx_e}{-z_e}, \quad y_p = \frac{ny_e}{-z_e}$$

此时, $x_p \in [l, r]$, $y_p \in [b, t]$,现将其映射到 $x_n \in [-1, 1], y_n \in [-1, 1]$

$$x_n = \frac{2}{r-l}x_p - \frac{r+l}{r-l} = (\frac{2n}{r-l}x_e + \frac{r+l}{r-l}z_e)/-z_e$$

$$y_n = \frac{2}{t-b}y_p - \frac{t+b}{t-b} = y_n = (\frac{2n}{t-b}y_e + \frac{t+b}{t-b}z_e)/-z_e$$

又由于 z_n 与 x_e, y_e 无关,则可以推断出

$$\mathbf{P} = \begin{pmatrix} \frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & \frac{t+b}{t-b} & 0 \\ * & * & A & B \\ 0 & 0 & -1 & 0 \end{pmatrix}$$

有性质 $w_c = -z_e$. 因为当 $z_e = -n$ 时, $z_n = -1$,当 $z_e = -f$ 时, $z_n = 1$.由

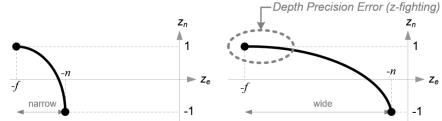
$$z_n = z_c/w_c = \frac{Az_e + B}{-z_e}$$

有

$$A = -\frac{f+n}{f-n}, \quad B = -\frac{2fn}{f-n}$$

$$\mathbf{P} = \begin{pmatrix} \frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & -\frac{f+n}{f-n} & -\frac{2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{pmatrix}$$

在近平面附近,深度精度较高,远平面附近深度精度较低,所以尽可能保证 z_e 与 z_f 间距离较小.



11.7 RTR约等式

一个重要的约等式

$$\int_{\Omega} f(x)g(x)dx \approx \frac{\int_{\Omega_G} f(x)dx}{\int_{\Omega_G} dx} \int_{\Omega} g(x)dx$$

约等号在

- Ω_G 是 g 的支集(函数值非零的自变量构成的集合)较小
- g 函数足够是光滑的.

满足两种条件中任一种时成立.

12 几何求交

12.1 射线求交

射线表达为

$$\mathbf{r}(t) = \mathbf{o} + t\mathbf{d}$$

线求交 设两条线分别为 $\mathbf{r}_1(t) = \mathbf{o}_1 + s\mathbf{d}_1, \mathbf{r}_2(t) = \mathbf{o}_2 + t\mathbf{d}_2$.

$$\mathbf{r}_1(s) = \mathbf{r}_2(t)$$

$$\begin{cases} s\mathbf{d}_1 \times \mathbf{d}_2 = (\mathbf{o}_2 - \mathbf{o}_1) \times \mathbf{d}_2 \\ t\mathbf{d}_2 \times \mathbf{d}_1 = (\mathbf{o}_1 - \mathbf{o}_2) \times \mathbf{d}_1 \end{cases}$$

$$\begin{cases} s(\mathbf{d}_1 \times \mathbf{d}_2) \cdot (\mathbf{d}_1 \times \mathbf{d}_2) = ((\mathbf{o}_2 - \mathbf{o}_1) \times \mathbf{d}_2) \cdot (\mathbf{d}_1 \times \mathbf{d}_2) \\ t(\mathbf{d}_2 \times \mathbf{d}_1) \cdot (\mathbf{d}_2 \times \mathbf{d}_1) = ((\mathbf{o}_1 - \mathbf{o}_2) \times \mathbf{d}_1) \cdot (\mathbf{d}_2 \times \mathbf{d}_1) \end{cases}$$

$$\begin{cases} s = \frac{\det(\mathbf{o}_2 - \mathbf{o}_1, \mathbf{d}_2, \mathbf{d}_1 \times \mathbf{d}_2)}{\|\mathbf{d}_1 \times \mathbf{d}_2\|^2} \\ t = \frac{\det(\mathbf{o}_2 - \mathbf{o}_1, \mathbf{d}_1, \mathbf{d}_1 \times \mathbf{d}_2)}{\|\mathbf{d}_1 \times \mathbf{d}_2\|^2} \end{cases}$$

射线与三角形求交(Moller Trumbore算法)

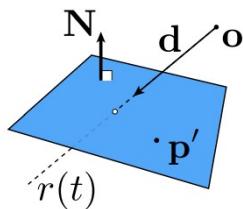
$$\mathbf{o} + t\mathbf{d} = (1 - b_1 - b_2)\mathbf{p}_0 + b_1\mathbf{p}_1 + b_2\mathbf{p}_2$$

$$\begin{pmatrix} t \\ b_1 \\ b_2 \end{pmatrix} = \frac{1}{\mathbf{s}_1 \cdot \mathbf{e}_1} \begin{pmatrix} \mathbf{s}_2 \cdot \mathbf{e}_2 \\ \mathbf{s}_1 \cdot \mathbf{s} \\ \mathbf{s}_2 \cdot \mathbf{d} \end{pmatrix}$$

$$\mathbf{e}_1 = \mathbf{p}_1 - \mathbf{p}_0, \quad \mathbf{e}_2 = \mathbf{p}_2 - \mathbf{p}_0, \quad \mathbf{s} = \mathbf{o} - \mathbf{p}_0$$

$$\mathbf{s}_1 = \mathbf{d} \times \mathbf{e}_2, \quad \mathbf{s}_2 = \mathbf{s} \times \mathbf{e}_1$$

射线与面相交



$$t = \frac{(\mathbf{p}' - \mathbf{o}) \cdot \mathbf{N}}{d \cdot \mathbf{N}}$$

射线与多边形相交 先判断射线与面相交,再判断交点是否在多边形内部.以下是2D中判断点 t 是否在多边形 $P = \{v_0, \dots, v_{n-1}\}$ 的伪代码.

```
bool PointInPolygon(t, P)
returns ({TRUE, FALSE});
1: bool inside = FALSE
2: e0 = v_{n-1}
3: bool y0 = (e0_y >= t_y)
4: for i = 0 to n - 1
5:   e1 = v_i
6:   bool y1 = (e1_y >= t_y)
7:   if(y0 != y1)
8:     if(((e1_y - t_y)(e0_x - e1_x) >= (e1_x - t_x)(e0_y - e1_y)) == y1)
9:       inside = !inside
10:  y0 = y1
11:  e0 = e1
12: return inside;
```

射线与球相交 球的曲面隐式表达

$$f(\mathbf{p}) = \|\mathbf{p} - \mathbf{c}\| - r = 0$$

由

$$f(\mathbf{r}(t)) = 0$$

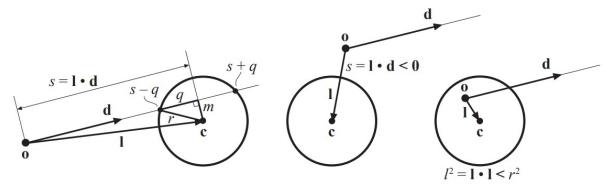
得

$$t^2 + 2t(\mathbf{d} \cdot (\mathbf{o} - \mathbf{c})) + (\mathbf{o} - \mathbf{c}) \cdot (\mathbf{o} - \mathbf{c}) - r^2 = 0$$

设 $b = \mathbf{d} \cdot (\mathbf{o} - \mathbf{c})$, $c = (\mathbf{o} - \mathbf{c}) \cdot (\mathbf{o} - \mathbf{c}) - r^2$, 当 $b^2 - c < 0$ 说明射线与球不相交,否则

$$t = -b \pm \sqrt{b^2 - c}$$

另外一个更优的方法

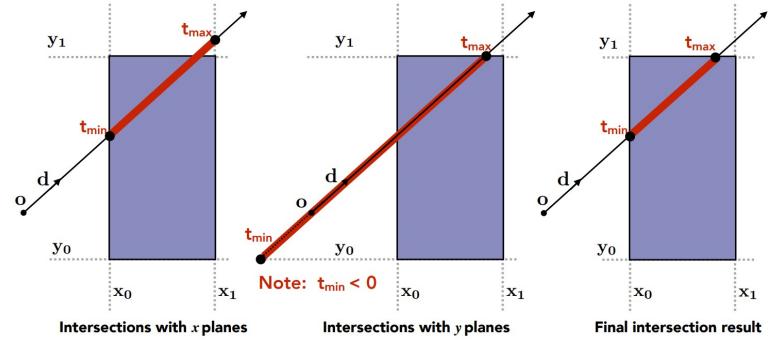


RaySphereIntersect($\mathbf{o}, \mathbf{d}, \mathbf{c}, r$)

returns ($\{\text{REJECT}, \text{INTERSECT}\}, t, \mathbf{p}$)

```
1: l = c - o
2: s = l * d
3: l^2 = l * l
4: if(s < 0 and l^2 > r^2) return (REJECT, 0, 0);
5: m^2 = l^2 - s^2
6: if(m^2 > r^2) return (REJECT, 0, 0);
7: q = sqrt(r^2 - m^2)
8: if(l^2 > r^2) t = s - q
9: else t = s + q
10: return (INTERSECT, t, o + td);
```

射线与AABB求交 以下是二维情况,对于三维情况简单扩展即可



关键点

- ray进入box当且仅当ray进入all pairs of slabs.
- ray离开box当且仅当ray离开any pairs of slabs.

当 $t_{\text{enter}} < t_{\text{exit}}$,则ray与AABB相交, $t_{\text{enter}} < 0$ 说明射线从AABB中发出,当 $t_{\text{exit}} < 0$ 则AABB与射线无相交.射线与AABB相交的条件为 $t_{\text{enter}} < t_{\text{exit}} \& t_{\text{exit}} \geq 0$.

求射线与 x 垂直的slabs相交的 t ,其余同理

$$t = \frac{\mathbf{p}'_x - \mathbf{o}_x}{d_x}$$

```
bool IntersectAABB(const Ray& ray, AABB& b) {
    const Vec& origin = ray.o, invDir = ray.d_inv;
    bool dirIsNeg[3] = { ray.d.x >= 0, ray.d.y >= 0, ray.d.z >= 0 };
    float tEnter = FLT_MIN, tExit = FLT_MAX;
    for (int i = 0; i < 3; i++) {
        float min = (b.p_min[i] - origin[i]) * invDir[i];
        float max = (b.p_max[i] - origin[i]) * invDir[i];
        if (!dirIsNeg[i]) swap(min, max);
        tEnter = std::max(min, tEnter);
        tExit = std::min(max, tExit);
    }
    return tEnter <= tExit && tExit >= 0;
}
```

射 线 与OBB求 交 其 中 \mathbf{a}^c 是 中 心 点,
 $\mathbf{a}^u, \mathbf{a}^v, \mathbf{a}^w$ 是OBB的normalized side directions, h_u, h_v, h_w 是
 从中心点到个面的长度.

```

RayOBBIntersect(o, d, A)
returns ({REJECT, INTERSECT}, t);
1 : tmin = -∞
2 : tmax = ∞
3 : p = ac - o
4 : for each i ∈ {u, v, w}
5 :   e = ai · p
6 :   f = ai · d
7 :   if(|f| > e)
8 :     t1 = (e + hi)/f
9 :     t2 = (e - hi)/f
10 :    if(t1 > t2) swap(t1, t2);
11 :    if(t1 > tmin) tmin = t1
12 :    if(t2 < tmax) tmax = t2
13 :    if(tmin > tmax) return (REJECT, 0);
14 :    if(tmax < 0) return (REJECT, 0);
15 :    else if(-e - hi > 0 or -e + hi < 0) return (REJECT, 0);
16 :    if(tmin > 0) return (INTERSECT, tmin);
17 :    else return (INTERSECT, tmax);
    
```

12.2 面求相交测试

三面交于一点 给定三个面的法向量 \mathbf{n}_i ,三个面上任意一点为 \mathbf{p}_i ,求三面交点 \mathbf{p}

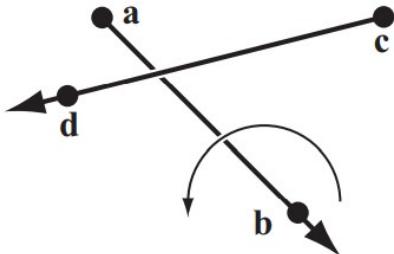
$$\mathbf{p} = \frac{(\mathbf{p}_1 \cdot \mathbf{n}_1)(\mathbf{n}_2 \times \mathbf{n}_3) + (\mathbf{p}_2 \cdot \mathbf{n}_2)(\mathbf{n}_3 \times \mathbf{n}_1) + (\mathbf{p}_3 \cdot \mathbf{n}_3)(\mathbf{n}_1 \times \mathbf{n}_2)}{\det(\mathbf{n}_1, \mathbf{n}_2, \mathbf{n}_3)}$$

如果只给定一个平面方程 $\mathbf{n}_i \mathbf{x} + d_i = 0$,需要得到面上任意一点,这里选择距离原点最近的点.

$$\begin{cases} \mathbf{r}_i(t) = t\mathbf{n}_i \\ \mathbf{n}_i \mathbf{x} + d_i = 0 \end{cases}$$

$$\mathbf{p}_i = -d_i \mathbf{n}_i$$

三角形面片与三角形面片相交



$$[\mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{d}] = (\mathbf{d} - \mathbf{a}) \cdot ((\mathbf{b} - \mathbf{a}) \times (\mathbf{c} - \mathbf{a}))$$

当 $[\mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{d}] = 0$ 说明 \mathbf{d} 在 $\triangle abc$ 所在平面上,当 $[\mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{d}] > 0$ 说明 \mathbf{d} 在 $\triangle abc$ 平面上方,否则在下方.

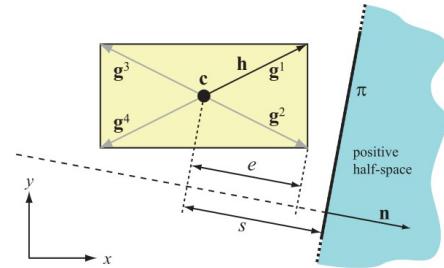
$T_1 = \triangle p_1 p_2 p_3, T_2 = \triangle q_1 q_2 q_3$, 分别在面 π_1, π_2 上,首先测试 T_1 是否与 π_2 相交, T_2 是否与 π_1 相交,如果不相交则 T_1, T_2 不会相交.

测试 $T_1 = \triangle p_1 p_2 p_3$ 是否与 π_1 相交,当 $[q_1, q_2, q_3, p_i], i = 1, 2, 3$ 同号且不为零的时候,不会相交.

π_1, π_2 相 交 于 线 L ,若 T_1, T_2 相 交,则 T_1, L 重 合 部 分 $L[i, j]$ 和 T_2, L 重合部分 $L[k, l]$ 要有重合,即 $k \leq j, i \leq l$ 等价于要求

$$[p_1, p_2, q_1, q_2] \leq 0 \& [p_1, p_3, q_1, q_2] \leq 0$$

面 与AABB相 交 测 试 假 设 有 AABB,两个 角 定 义 为 $\mathbf{b}_{max}, \mathbf{b}_{min}$, 中 心 为 $\mathbf{c} = (\mathbf{b}_{max} + \mathbf{b}_{min})/2$, positive half diagonal vector是 $\mathbf{h} = (\mathbf{b}_{max} - \mathbf{b}_{min})/2$,待 测 试 面 为 $\mathbf{n} \cdot \mathbf{x} + d = 0$



```

PlaneAABBIntersect(B, π)
returns ({OUTSIDE, INSIDE, INTERSECTING});
1 : c = (bmax + bmin)/2
2 : h = (bmax - bmin)/2
3 : e = hx|nx| + hy|ny| + hz|nz|
4 : s = c · n + d
5 : if(s - e > 0) return (OUTSIDE);
6 : if(s + e < 0) return (INSIDE);
7 : return (INTERSECTING);
    
```

面 与OBB相 交 测 试 $(\mathbf{b}_u, \mathbf{b}_v, \mathbf{b}_w)$ 是 coordinate system axes, (h_u^B, h_v^B, h_w^B) 是 box 在 axes 上 的 投 影 长 度.与 面 与AABB相 交 测 试 不 同 的 是

$$e = h_u^B |\mathbf{n} \cdot \mathbf{b}_u| + h_v^B |\mathbf{n} \cdot \mathbf{b}_v| + h_w^B |\mathbf{n} \cdot \mathbf{b}_w|$$

三 角 形 和AABB相 交 不 失 一 般 性AABB中 心 为 原 点, vector of half lengths是 \mathbf{h} , 三 角 形 为 $\triangle v_0 v_1 v_2$, 设 边 向 量 分 别 为 $\mathbf{f}_0 = \mathbf{v}_1 - \mathbf{v}_0, \mathbf{f}_1 = \mathbf{v}_2 - \mathbf{v}_1$ 和 $\mathbf{f}_2 = \mathbf{v}_0 - \mathbf{v}_2, \mathbf{e}_0 = (1, 0, 0), \mathbf{e}_1 = (0, 1, 0), \mathbf{e}_2 = (0, 0, 1)$

$$\mathbf{a}_{ij} = \mathbf{e}_i \times \mathbf{f}_j, i, j \in \{0, 1, 2\}$$

将 三 角 形 上 的 点 投 射 到 \mathbf{a} 上 去(成 为 一 根 线)

$$p_k = \mathbf{a}_{ij} \cdot \mathbf{v}_k, k \in \{0, 1, 2\}$$

有 性 质 $p_0 = p_1$, 找 到 $p_k, k \in \{0, 1, 2\}$ 中 的 最 大 值 $\max(p_0, p_2)$ 和 最 小 值 $\min(p_0, p_2)$ 即 可 获 得 三 角 形 在 \mathbf{a} 上 投 影 的 线. 再 将 box 投 影 到 \mathbf{a} 上 去, 记 半 径 r

$$r = h_x|a_x| + h_y|a_y| + h_z|a_z| = h_y|a_y| + h_z|a_z|$$

最终在axis上的test变为

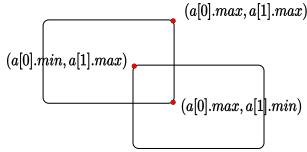
$if(\min(p_0, p_2) > r \text{ or } \max(p_0, p_2) < -r) \text{ return false}$

还有很多算法能解决任意多边形和box的相交测试,三角形与球体相交测试等.

12.3 体的相交测试

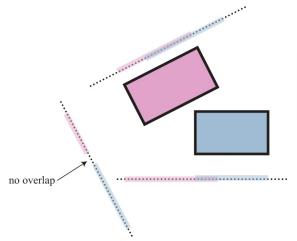
AABB相交测试 判断两个三维AABB是否相交十分简单,假设两个三维AABB是 a, b , a 在第x轴上最大坐标值表示为 $a.\max[0]$.

```
bool test2AABB(const AABB& a, const AABB& b){
    if (a.max[0] < b.min[0] || a.min[0] > b.max[0])
        return false;
    if (a.max[1] < b.min[1] || a.min[1] > b.max[1])
        return false;
    if (a.max[2] < b.min[2] || a.min[2] > b.max[2])
        return false;
    return true;
}
```



OBB相交测试 类似三角形与OBB相交测试,三维情况在15个axes上做相交测试,如果在一个axes上投影没有overlap则OBB不相交.

设两个OBB分别是 A, B , A 的轴为 \mathbf{a}_i , B 的轴为 \mathbf{b}_i ,15个axes分别是 $\mathbf{a}_i, \mathbf{b}_i, \mathbf{a}_i \times \mathbf{b}_j, \forall i, j \in x, y, z$



12.4 View Frustum相交测试

提取Frustum的平面 设三维上某点为 s ,view matrix为 V ,投影矩阵为 P , $M = PV$, $t = Ms$,设 $u = t/t_w$.只有当 $-1 \leq u_i \leq 1$ 时, s 在Frustum里面.

$$-1 \leq u_x \Leftrightarrow -1 \leq \frac{t_x}{t_w} \Leftrightarrow t_x + t_w \geq 0$$

$$\mathbf{m}_0 \cdot \mathbf{s} + \mathbf{m}_3 \cdot \mathbf{s} \geq \Leftrightarrow \mathbf{m}_0 + \mathbf{m}_3 \cdot \mathbf{s} \geq 0$$

同理可以推出包围Frustum的平面方程.

$$\begin{aligned} -(\mathbf{m}_3 + \mathbf{m}_0) \cdot (x, y, z, 1) &= 0 & [\text{left}], \\ -(\mathbf{m}_3 - \mathbf{m}_0) \cdot (x, y, z, 1) &= 0 & [\text{right}], \\ -(\mathbf{m}_3 + \mathbf{m}_1) \cdot (x, y, z, 1) &= 0 & [\text{bottom}], \\ -(\mathbf{m}_3 - \mathbf{m}_1) \cdot (x, y, z, 1) &= 0 & [\text{top}], \\ -(\mathbf{m}_3 + \mathbf{m}_2) \cdot (x, y, z, 1) &= 0 & [\text{near}], \\ -(\mathbf{m}_3 - \mathbf{m}_2) \cdot (x, y, z, 1) &= 0 & [\text{far}]. \end{aligned}$$

做Frustum与其他几何体相交测试可以退化成6个面分别与几何体相交测试.

13 球谐函数

球面坐标可以表似为

$$x = r\sin\theta\cos\phi \quad y = r\sin\theta\sin\phi \quad z = r\cos\theta$$

空间中laplace 方程可以表示为

$$\nabla^2 = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} + \frac{\partial^2}{\partial z^2} = 0$$

球面坐标代入Laplace方程

$$\frac{1}{r^2} \frac{\partial}{\partial r} (r^2 \frac{\partial f}{\partial r}) + \frac{1}{r^2 \sin\theta} \frac{\partial}{\partial \theta} (\sin\theta \frac{\partial f}{\partial \theta}) + \frac{1}{r^2 \sin^2\theta} \frac{\partial^2 f}{\partial \phi^2} = 0$$

设 $f(r, \theta, \phi) = R(r)Y(\theta, \phi)$

$$\frac{Y}{r^2} \frac{\partial}{\partial r} (r^2 \frac{\partial R}{\partial r}) + \frac{R}{r^2 \sin\theta} \frac{\partial}{\partial \theta} (\sin\theta \frac{\partial Y}{\partial \theta}) + \frac{R}{r^2 \sin^2\theta} \frac{\partial^2 Y}{\partial \phi^2} = 0$$

移项,发现左右两边相等仅可能两边是同一个常数

$$\frac{1}{R} \frac{\partial}{\partial r} (r^2 \frac{\partial R}{\partial r}) = -\frac{1}{Y \sin\theta} \frac{\partial}{\partial \theta} (\sin\theta \frac{\partial Y}{\partial \theta}) - \frac{1}{Y \sin^2\theta} \frac{\partial^2 Y}{\partial \phi^2} = l(l+1)$$

分别解方程

$$\begin{aligned} \frac{\partial}{\partial r} (r^2 \frac{\partial R}{\partial r}) - l(l+1)R &= 0 \\ \frac{1}{\sin\theta} \frac{\partial}{\partial \theta} (\sin\theta \frac{\partial Y}{\partial \theta}) + \frac{1}{\sin^2\theta} \frac{\partial^2 Y}{\partial \phi^2} + l(l+1)Y &= 0 \end{aligned}$$

只关心角度部分的解,即后一个方程的解,也称为球函数方程

$$Y(\theta, \phi) = \Theta(\theta)\Phi(\phi)$$

带入球函数

$$\frac{\Phi}{\sin\theta} \frac{\partial}{\partial \theta} (\sin\theta \frac{\partial \Theta}{\partial \theta}) + \frac{\Theta}{\sin^2\theta} \frac{\partial^2 \Phi}{\partial \phi^2} + l(l+1)\Theta\Phi = 0$$

移项,发现左右两边相等仅可能两边是同一个常数

$$\frac{\sin\theta}{\Theta} \frac{\partial}{\partial \theta} (\sin\theta \frac{\partial \Theta}{\partial \theta}) + l(l+1)\sin^2\theta = -\frac{1}{\Phi} \frac{\partial^2 \Phi}{\partial \phi^2} = \lambda$$

分解两个常微分方程

$$\sin\theta \frac{\partial}{\partial \theta} (\sin\theta \frac{\partial \Theta}{\partial \theta}) + (l(l+1)\sin^2\theta - \lambda)\Theta = 0$$

$$\frac{\partial^2 \Phi}{\partial \phi^2} + \lambda \Phi = 0$$

由 $\Phi(\phi + 2\pi) = \Phi(\phi)$, $\sin m\phi + i \cos m\phi = e^{im\phi}$,解得

$$\Phi(\phi) = e^{im\phi}, m = 0, \pm 1, \pm 2, \dots$$

设 $x = \cos\theta$

$$\begin{aligned} \frac{1}{\sin\theta} \frac{\partial}{\partial\theta} (\sin\theta \frac{\partial\Theta}{\partial\theta}) + (l(l+1) - \frac{m^2}{\sin^2\theta})\Theta &= 0 \\ \frac{\partial\Theta}{\partial\theta} &= \frac{\partial\Theta}{\partial x} \frac{\partial x}{\partial\theta} = -\sin\theta \frac{\partial\Theta}{\partial x} \\ (1-x^2) \frac{\partial^2\Theta}{\partial x^2} - 2x \frac{\partial\Theta}{\partial x} \frac{\partial\Theta}{\partial x} + l(l+1)\Theta &= 0 \end{aligned}$$

这个方程式l次连带勒让德方程,或缩合勒让德方程,解称为连带勒让德函数.当 $\lambda = l(l+1), l = 0, 1, \dots$ 时才有有界周期解,用 $P_l^m(x)$ 表示

$$\Theta(\theta) = P_l^m(\cos\theta), m = 0, \pm 1, \pm 2, \dots, \pm l$$

勒让德函数为,也叫l次m阶连带勒让德函数.

$$P_l^m(x) = \frac{(-1)^m (1-x^2)^{\frac{m}{2}}}{2^l l!} \frac{d^{l+m}}{dx^{l+m}} (x^2 - 1)^l$$

勒让德函数递归关系(便于计算机实现)

$$\left\{ \begin{array}{l} (l-m)P_l^m(x) = x(2l-1)P_{l-1}^m(x) - (l+m-1)P_{l-2}^m(x) \\ P_m^m(x) = (-1)^m (2m-1)!! (1-x^2)^{m/2} \\ P_{m+1}^m(x) = x(2m+1)P_m^m(x) \end{array} \right.$$

其中!!表示双阶乘, $(2m-1)!! = 1 \cdot 3 \cdot 5 \dots (2m-1)$

所以

$$Y(\theta, \phi) = \sum_{l=0}^{\infty} \sum_{k=-l}^l P_l^k(\cos\theta) e^{im\phi}, m = 0, \pm 1, \pm 2, \dots$$

球谐函数的模长

$$(N_l^m)^2 = \int_S Y_{lm}(x) Y_{lm}^*(x) \sin\theta d\theta d\phi = \frac{4\pi}{2l+1} \frac{(l+|m|)!}{(l-|m|)!}$$

$Y_{lm}(\theta, \phi)$ 表示一般形式的球谐函数, $Y_l^m(\theta, \phi)$ 表示归一化球谐函数

$$\begin{aligned} Y_l^m(\theta, \phi) &= K_l^m Y_{lm}(\theta, \phi) \\ K_l^m &= \frac{1}{N_l^m} = \sqrt{\frac{2l+1}{4\pi} \frac{(l-|m|)!}{(l+|m|)!}} \\ Y_l^m &= \begin{cases} \sqrt{2} K_l^m \cos(m\phi) P_l^m(\cos\theta), & m > 0 \\ \sqrt{2} K_l^m \sin(-m\phi) P_l^{-m}(\cos\theta), & m < 0 \\ K_l^0 P_l^m(\cos\theta), & m = 0 \end{cases} \end{aligned}$$

	$m = -3$	$m = -2$	$m = -1$	$m = 0$	$m = 1$	$m = 2$	$m = 3$
0				$\frac{1}{2}\sqrt{\frac{1}{2}}y$			
1			$-\frac{1}{2}\sqrt{\frac{3}{2}}y$	$\frac{1}{2}\sqrt{\frac{3}{2}}z$	$-\frac{1}{2}\sqrt{\frac{3}{2}}x$		
2		$\frac{1}{2}\sqrt{\frac{15}{2}}yx$	$-\frac{1}{2}\sqrt{\frac{15}{2}}yz$	$\frac{1}{4}\sqrt{\frac{5}{2}}(3z^2 - 1)$	$-\frac{1}{2}\sqrt{\frac{15}{2}}zx$	$\frac{1}{4}\sqrt{\frac{15}{2}}(x^2 - y^2)$	
3	$-\frac{1}{8}\sqrt{\frac{105}{2}}y(3x^2 - y^2)$	$\frac{1}{2}\sqrt{\frac{105}{2}}xyz$	$-\frac{1}{8}\sqrt{\frac{105}{2}}y(3z^2 - 1)$	$\frac{1}{4}\sqrt{\frac{5}{2}}(5z^2 - 3)$	$-\frac{1}{8}\sqrt{\frac{105}{2}}z(5z^2 - 1)$	$\frac{1}{4}\sqrt{\frac{105}{2}}z(x^2 - y^2)$	$-\frac{1}{8}\sqrt{\frac{105}{2}}z(x^2 - 3y^2)$

14 随机采样

14.1 采样方法

问题:已知随机变量 Z 的概率密度函数 $p(z)$,求按分布随机生成 z 的算法.

逆变换采样(The Inverse Transform Method) 已知pdf,先求累积分布函数cdf.

$$F(z) = \int p(z) dz$$

$F(z)$ 的值域为 $[0, 1]$,计算机生成随机变量 $\varepsilon \sim uniform(0, 1)$,然后由

$$F(z) = \varepsilon \Rightarrow z = F^{-1}(\varepsilon)$$

即可根据 ε 求出符合分布的一个样本 z .

对于多维随机变量 \mathbf{z} ,一般问题中,各维度随机变量是独立分布的,故,各个维度参数 z_i 的cdf可以写为

$$F(\mathbf{z}_i) = \int \dots \int \int \dots \int F(\mathbf{z}) dz_0 \dots dz_{i-1} dz_{i+1} \dots dz_n$$

这种方法对于复杂问题难以从pdf推出cdf.

拒绝采样 对于概率分布 $p(\mathbf{z})$,如果cdf很难求,则引入一个简单的提议分布 $q(\mathbf{z})$ (易求其cdf),使得

$$\forall \mathbf{z}_i, M q(\mathbf{z}_i) \geq q(\mathbf{z}_i)$$

定义接受率

$$\alpha = \frac{p(\mathbf{z}_i)}{M q(\mathbf{z}_i)}$$

算法过程如下:

1. 取 $\mathbf{z}_i \sim q(\mathbf{z})$.
2. 取 $u \sim uniform(0, 1)$
3. 如果 $u \leq \alpha$, 则接受 \mathbf{z}_i ,否则,拒绝这个值.

对于一些高维的复杂的分布 $p(\mathbf{z})$ 难以找到一个合适的 $q(\mathbf{z})$ 和 M .

马尔科夫链蒙特卡洛(MCMC) 假设已知连续随机变量 \mathbf{X} 的概率分布 $p(\mathbf{x})$.

马尔科夫链是一种时间和状态都是离散的随机变量序列,齐次一阶马尔科夫链满足:

$$p(\mathbf{x}_{t+1} | \mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_t) = p(\mathbf{x}_{t+1} | \mathbf{x}_t)$$

即某时刻的状态转移概率值依赖于前一个状态,只要知道了任意两个状态转移概率,马尔科夫链模型就定了.

性质：设马尔可夫链转移矩阵为 $\mathbf{P}_{ij} = p(\mathbf{X}_{t+1} = j | \mathbf{X}_t = i)$, 对于任意初始概率分布 $\boldsymbol{\pi}_i^{(0)} = p(\mathbf{X} = i)$

$$\boldsymbol{\pi}^{t+1} = \mathbf{P}\boldsymbol{\pi}^{(t)} \Rightarrow p(\mathbf{x}_{t+1}) = \int p(\mathbf{x}_{t+1} | \mathbf{x}_t) p(\mathbf{x}_t) d\mathbf{x}_t$$

$$\boldsymbol{\pi}^{(n)} = \mathbf{P}^n \boldsymbol{\pi}^{(0)} \Rightarrow$$

$$p(\mathbf{x}_n) = \int p(\mathbf{x}_n | \mathbf{x}_{n-1}) \cdots \int p(\mathbf{x}_2 | \mathbf{x}_1) \int p(\mathbf{x}_1 | \mathbf{x}_0) p(\mathbf{x}_0) d\mathbf{x}_0 d\mathbf{x}_1 \dots d\mathbf{x}_{n-1}$$

当 n 足够大时, 概率分布 $\boldsymbol{\pi}^{(n)}$ 不再发生变化. 此外, \mathbf{P}^n 的值稳定, 即每一行相等, 且都等于 $\boldsymbol{\pi}^{(n)}$.

$$\lim_{n \rightarrow \infty} \mathbf{P}_{ij}^n = \boldsymbol{\pi}_j^{(n)} = \boldsymbol{\pi}_j^*$$

$$\boldsymbol{\pi}_j^* = \sum_{i=0}^{\infty} \boldsymbol{\pi}_i^* \mathbf{P}_{ij} \Rightarrow \boldsymbol{\pi}^* = \mathbf{P} \boldsymbol{\pi}^*$$

以下概述马尔可夫链采样过程

1. 给定马尔科夫链状态转移矩阵 \mathbf{P} , 设置状态转移次数阈值 n_1 , 需要的样本数 n_2 .
2. 从任意简单概率分布(如高斯分布)采样得到初始状态分布 x_0 .
3. 对于 $t \in [0, n_1 + n_2 - 1]$, \mathcal{Z}^+ , 从条件概率分布 $P(x|x_t)$ 中采样得到样本 x_{t+1} .
4. 样本集 $(x_{n_1}, \dots, x_{n_1+n_2-1})$ 为所求样本集.

给定一个概率平稳分布 $\boldsymbol{\pi}$, 难以直接找到对应的状态转移矩阵 \mathbf{P} 马尔可夫链细致平稳条件, 定义: 若非周期马尔科夫链状态转移矩阵 \mathbf{P} 和状态分布 $\boldsymbol{\pi}$ 对于所有的 i, j 满足

$$\boldsymbol{\pi}_i \mathbf{P}_{ij} = \boldsymbol{\pi}_j \mathbf{P}_{ji}$$

则概率分布 $\boldsymbol{\pi}$ 是状态转移矩阵 \mathbf{P} 的平稳分布

可推出

$$\sum_{i=1}^{\infty} \boldsymbol{\pi}_i \mathbf{P}_{ij} = \sum_{i=1}^{\infty} \boldsymbol{\pi}_j \mathbf{P}_{ji} = \boldsymbol{\pi}_j \sum_{i=1}^{\infty} \mathbf{P}_{ji} = \boldsymbol{\pi}_j$$

可矩阵表达如下,

$$\boldsymbol{\pi} \mathbf{P} = \boldsymbol{\pi}$$

由此可见, 让 $\boldsymbol{\pi}$ 满足细致平稳分布的 \mathbf{P} 满足马尔可夫收敛性, 只要找到可以使概率分布 $\boldsymbol{\pi}$ 满足细致平稳分布的矩阵 \mathbf{P} 即可.

MCMC采样, 对于目标平稳分布 $\boldsymbol{\pi}$ 和任意一个马尔科夫链状态转移矩阵不满足细致平衡条件, 即

$$\boldsymbol{\pi}_i \mathbf{Q}_{ij} \neq \boldsymbol{\pi}_j \mathbf{Q}_{ji}$$

设

$$\boldsymbol{\alpha}_{ij} = \boldsymbol{\pi}_j \mathbf{Q}_{ji}, \quad \boldsymbol{\alpha}_{ji} = \boldsymbol{\pi}_i \mathbf{Q}_{ij}$$

这样可以得到分布 $\boldsymbol{\pi}$ 的马尔可夫链状态转移矩阵

$$\mathbf{P}_{i,j} = \mathbf{Q}_{i,j} \boldsymbol{\alpha}_{i,j}$$

1. 给定马尔科夫链状态转移矩阵 \mathbf{P} , 设置状态转移次数阈值 n_1 , 需要的样本数 n_2 .
2. 从任意简单概率分布得到初始状态 \mathbf{x}_0 .
3. 对于 $t \in [0, n_1 + n_2 - 1]$, \mathcal{Z}^+
 - (a) 从条件概率分布 $Q(\mathbf{x}|\mathbf{x}_t)$ (可取高斯分布)中采样得到样本 \mathbf{x}_* .
 - (b) 从均匀分布采样 $u \sim uniform[0, 1]$
 - (c) 如果 $u \leq \alpha(\mathbf{x}_t, \mathbf{x}_*) = \boldsymbol{\pi}(\mathbf{x}_*) Q(\mathbf{x}_* | \mathbf{x}_t)$, 则 $\mathbf{x}_{t+1} = \mathbf{x}_*$, 否则 $\mathbf{x}_{t+1} = \mathbf{x}_t$
4. 样本集 $(\mathbf{x}_{n_1}, \dots, \mathbf{x}_{n_1+n_2-1})$ 为所求样本集.

Metropolis-Hastings采样, MCMC采样中, 由于 $\alpha(\mathbf{x}_t, \mathbf{x}_*)$ 可能会非常小, 导致大部分采样值被拒绝转移了, 采样效率低, M-H采样改进这个问题. $\alpha(\mathbf{x}_t, \mathbf{x}_*)$ 对应 α_{ij} , 再MCMC采样细致平稳条件下

$$\boldsymbol{\pi}_i \mathbf{Q}_{ij} \boldsymbol{\alpha}_{ij} = \boldsymbol{\pi}_j \mathbf{Q}_{ji} \boldsymbol{\alpha}_{ji}$$

故可取

$$\alpha_{ij} = \min\left\{\frac{\boldsymbol{\pi}_j \mathbf{Q}_{ji}}{\boldsymbol{\pi}_i \mathbf{Q}_{ij}}, 1\right\}$$

很多时候选择 \mathbf{Q} 为对称的, 即 $\mathbf{Q}_{ij} = \mathbf{Q}_{ji}$

$$\alpha_{ij} = \min\left\{\frac{\boldsymbol{\pi}_j}{\boldsymbol{\pi}_i}, 1\right\}$$

MH的缺点, 需要计算接受率, 所以在高维时计算量大, 并且由于接受率的原因导致算法收敛时间变长. 有些高位数据, 特征条件概率分布好求, 但是特征联合分布不好求, 因此提出了Gibbs采样解决这个问题.

14.2 在流形上均匀采样的通用做法

基本问题: 已知参数空间 \mathcal{R}^d , n 维流形 \mathcal{M}^n 为和一个映射 $\phi : \mathcal{R}^d \rightarrow \mathcal{M}^n$, 问: 如何在生成样本使其在 \mathcal{M}^n 上均匀分布.

解决思路: 设随机变量 $\mathbf{x} \in \mathcal{R}^d$, 求 $p(\mathbf{x})$ 令 $\phi(\mathbf{x}) \in \mathcal{M}^n$ 时均匀分布,

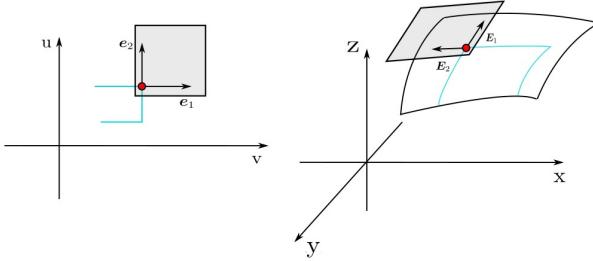
设参数空间中与 \mathbf{x} 对应的微小的 d 维体积 $dV_{\mathbf{x}}$ 和概率密度 $p(\mathbf{x})$, 在流形 \mathcal{M} 上找到 $\phi(\mathbf{x})$ 和对应的 d 维体积 $dV_{\phi(\mathbf{x})}$ 和概率密度 $p(\phi(\mathbf{x}))$.

$$p(\mathbf{x}) dV_{\mathbf{x}} = p(\phi(\mathbf{x})) dV_{\phi(\mathbf{x})}$$

因为要在 \mathcal{M}^d 上均匀采样, $p(\phi(\mathbf{x})) = \frac{1}{V_M}$, 其中 V_M 是整个子流形的 d 维总体积.

$$p(\mathbf{x}) = \frac{dV_{\phi(\mathbf{x})}}{dV_{\mathbf{x}}} p(\phi(\mathbf{x})) = \frac{dV_{\phi(\mathbf{x})}}{dV_{\mathbf{x}} V_M}$$

以下讨论求 $\frac{dV_{\phi(x)}}{dV_x}$, 设在 x 处局部正交单位坐标系 由于 θ, φ 相互独立
为 e_1, \dots, e_d , 其在 $\phi(x)$ 处切向量为 E_1, \dots, E_d



$$E_i = \frac{\partial \phi}{\partial x_i}, i = 1, \dots, d$$

引入Gram行列式, 若 $V = (v_1, \dots, v_k)$, Gram行列式为

$$G(V) = \det(V^T V)$$

其几何意义为向量 v_1, \dots, v_k 张出的空间体体积平方, 即这些向量的外积. 所以

$$\frac{dV_{\phi(x)}}{dV_x} = \sqrt{\det\left(\frac{\partial \phi^T}{\partial x} \frac{\partial \phi}{\partial x}\right)}$$

故

$$p(x) = \frac{1}{V_M} \sqrt{\det\left(\frac{\partial \phi^T}{\partial x} \frac{\partial \phi}{\partial x}\right)}$$

计算机实现时, 对于每个采样点, 先生成 $\epsilon \sim uniform[0, 1]$, 然后根据逆变换采样得到的表达式求出 $x = F^{-1}(\epsilon)$, 最后得到流形上的一个采样点 $\phi(x)$.

球面上的均匀采样

$$\phi(x) = \phi(\theta, \varphi) = \begin{pmatrix} r \sin \theta \cos \varphi \\ r \sin \theta \sin \varphi \\ r \cos \theta \end{pmatrix}, \theta \in (0, \pi), \varphi \in (0, 2\pi)$$

$$\frac{\partial \phi}{\partial x} = \begin{pmatrix} \frac{\partial \phi_x}{\partial \theta} & \frac{\partial \phi_x}{\partial \varphi} \\ \frac{\partial \phi_y}{\partial \theta} & \frac{\partial \phi_y}{\partial \varphi} \\ \frac{\partial \phi_z}{\partial \theta} & \frac{\partial \phi_z}{\partial \varphi} \end{pmatrix} = \begin{pmatrix} r \cos \theta \cos \varphi & -r \sin \theta \sin \varphi \\ r \cos \theta \sin \varphi & -r \sin \theta \cos \varphi \\ -r \sin \theta & 0 \end{pmatrix}$$

$$\frac{\partial \phi^T}{\partial x} \frac{\partial \phi}{\partial x} = \begin{pmatrix} r^2 & \\ & r^2 \sin^2 \theta \end{pmatrix}$$

$$\sqrt{\det\left(\frac{\partial \phi^T}{\partial x} \frac{\partial \phi}{\partial x}\right)} = r^2 \sin \theta$$

易得球面面积为 $4\pi r^2$, 因此

$$p(\theta, \varphi) = \frac{\sin \theta}{4\pi}$$

$$p(\theta) = \int_0^{2\pi} \frac{\sin \theta}{4\pi} d\varphi = \frac{\sin \theta}{2}$$

$$p(\varphi) = \int_0^\pi \frac{\sin \theta}{4\pi} d\varphi = \frac{1}{2\pi}$$

使用逆变采样, 两个变量的CDF分别为

$$F(\theta) = \int_0^\theta p(\theta) d\theta = \frac{1 - \cos \theta}{2}$$

$$F(\varphi) = \int_0^\varphi p(\varphi) d\varphi = \frac{\varphi}{2\pi}$$

设 $\varepsilon_1, \varepsilon_2 \sim uniform[0, 1]$

$$\varepsilon_1 = \frac{1 - \cos \theta}{2}$$

$$\varepsilon_2 = \frac{\varphi}{2\pi}$$

所以

$$\theta = \arccos(1 - 2\varepsilon_1)$$

$$\varphi = 2\pi\varepsilon_2$$

带入到参数方程可得

$$x = 2r \cos(2\pi\varepsilon_2) \sqrt{\varepsilon_1(1 - \varepsilon_1)}$$

$$y = 2r \sin(2\pi\varepsilon_2) \sqrt{\varepsilon_1(1 - \varepsilon_1)}$$

$$z = r(1 - 2\varepsilon_1)$$

三角形上均匀采样 记三角形三个顶点为 A, B, C , 面积为 S

$$x = (u, v), \quad \phi(x) = uA + vB + (1 - u - v)C$$

好吧

$$\frac{\partial \phi}{\partial x} = (\overrightarrow{CA} \quad \overrightarrow{CB})$$

设 $\overrightarrow{CA}, \overrightarrow{CB}, \overrightarrow{AB}$ 对应的三角形三边为 a, b, c .

$$\det\left(\frac{\partial \phi^T}{\partial x} \frac{\partial \phi}{\partial x}\right) = (\overrightarrow{CA} \overrightarrow{CA})(\overrightarrow{CB} \overrightarrow{CB}) - (\overrightarrow{CA} \overrightarrow{CB})^2$$

$$= b^2 a^2 - |ab \cos C|^2 = a^2 b^2 \sin(C)^2 = 4\left(\frac{1}{2}ab \sin C\right)^2 =$$

$$p(u, v) = \frac{2S}{S} = 2, \quad u > 0, \quad v > 0, \quad u + v < 1$$

注意此处, u, v 两个随机变量不独立.

$$p(u) = \int_0^{1-u} 2dv = 2(1 - u), \quad u \in (0, 1)$$

$$F(u) = 2u - u^2, \quad u \in (0, 1)$$

设 $\varepsilon_1, \varepsilon_2 \sim uniform(0, 1)$

$$2u - u^2 = \varepsilon_1$$

$$u = 1 - \sqrt{1 - \varepsilon_1}$$

因为 $1 - \varepsilon_1$ 同样是一个 $[0, 1]$ 上的均匀分布, 所以

$$u = 1 - \sqrt{\varepsilon_1}$$

$$p(v|u) = \frac{p(u, v)}{p(u)} = \frac{1}{1-u}$$

$$F(v|u) = \frac{v}{1-u} = \varepsilon_2$$

$$v = \sqrt{\varepsilon_1} \varepsilon_2$$

故在三角形上的均匀采样表达为

$$\phi = (1 - \sqrt{\varepsilon_1})\mathbf{A} + (\sqrt{\varepsilon_1}\varepsilon_2)\mathbf{B} + (\sqrt{\varepsilon_1}(1 - \varepsilon_2))\mathbf{C}$$