UE4的GTAO简析

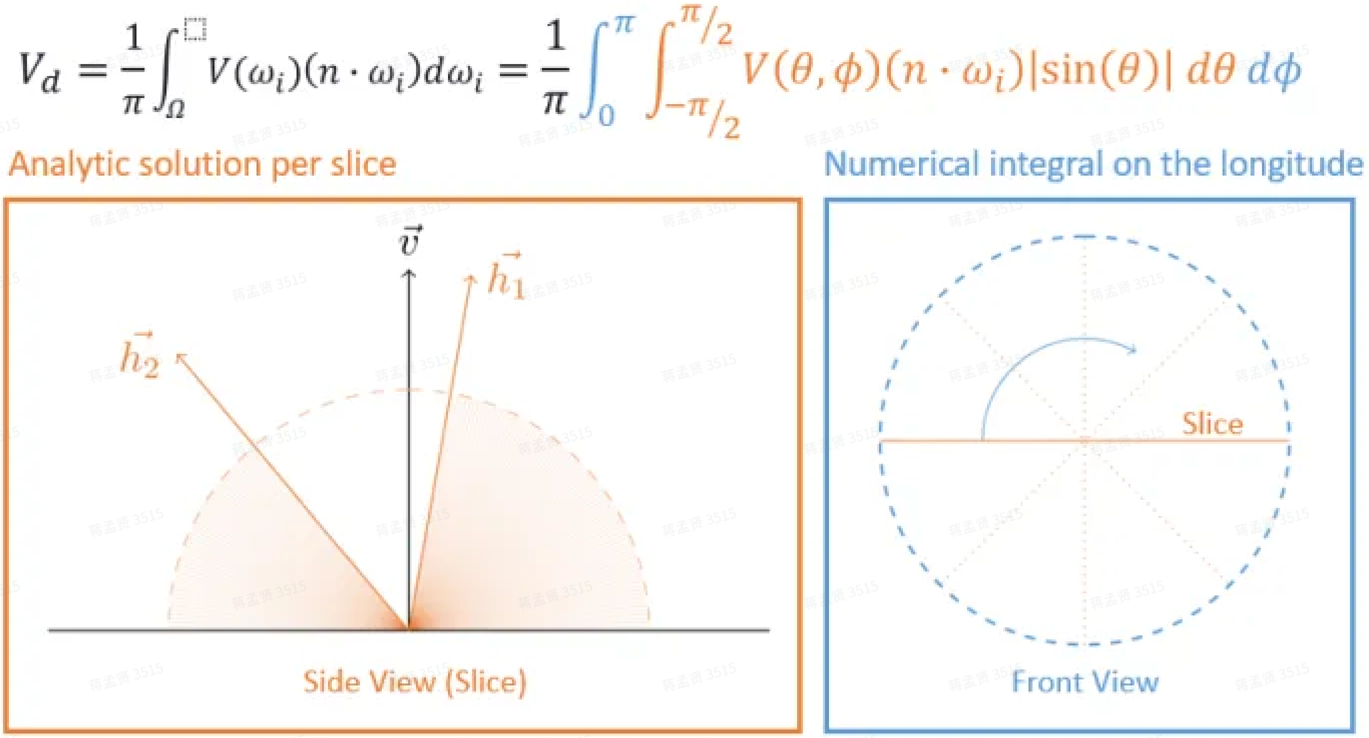
1. 原论文公式推导

$$AO = 1 - \frac{1}{\pi}\int_{\Omega} V(\vec{w}_i)(\vec{n}\cdot\vec{w}_i)dw_i$$

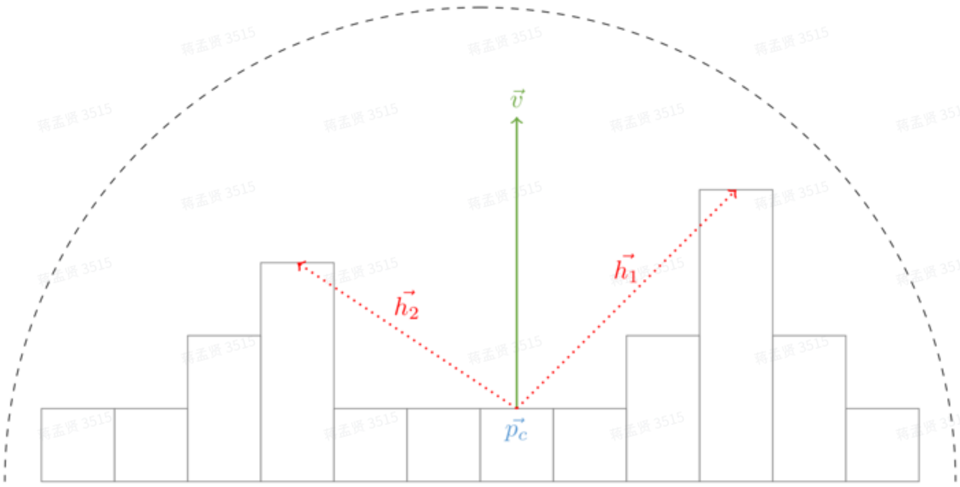
$$= 1 - \frac{1}{\pi}\int_0^{\pi}\int_{-\frac{\pi}{2}}^{\frac{\pi}{2}} V(\theta,\phi)(\vec{n}\cdot\vec{w}_i)|sin(\theta)|d\theta\, d\phi$$

其中:

- $\vec{n}$ 是法线
- $V(\cdot)$ 是可见性函数，被遮挡返回 `1`，不遮挡返回 `0`
- θ ：垂直角/天顶角
- ϕ ：水平角



GTAO应用了一种切面思想：AO的大小实际上和有效积分区域有关，上左图中的浅红区域就是有效积分区域。每一个 ϕ 角对应了一个切片，而切片的有效积分区域和 $(\vec{h}_1, \vec{h}_2)$ 有关，而 $(\vec{h}_1, \vec{h}_2)$ 则怎么来的？实际上就是HBAO中最大垂直角 α 。



Side View (Slice)

成对垂直角的计算

`GTAO` 中采取了不同的计算方式，这个或许和它的积分方式有关？实现起来很简单：

a. 随机采样N个成对的方向 $(\vec{w}_1, \vec{w}_2)$ ——每个成对的方向都是一个切片，也就是一个 ϕ 角。这个随机过程和HBAO共用代码

b. 对其中一个方向对，进行M次像素采样，记其中一对采样为 $(p1, p2)$

c. 使用 `ScreenToViewPos` 方法获得其ViewSpace的位置 $(pos1, pos2)$

d. 使用 $ViewPos$ 和 $(pos1, pos2)$ 相减，获得 $(\vec{h}_1, \vec{h}_2)$

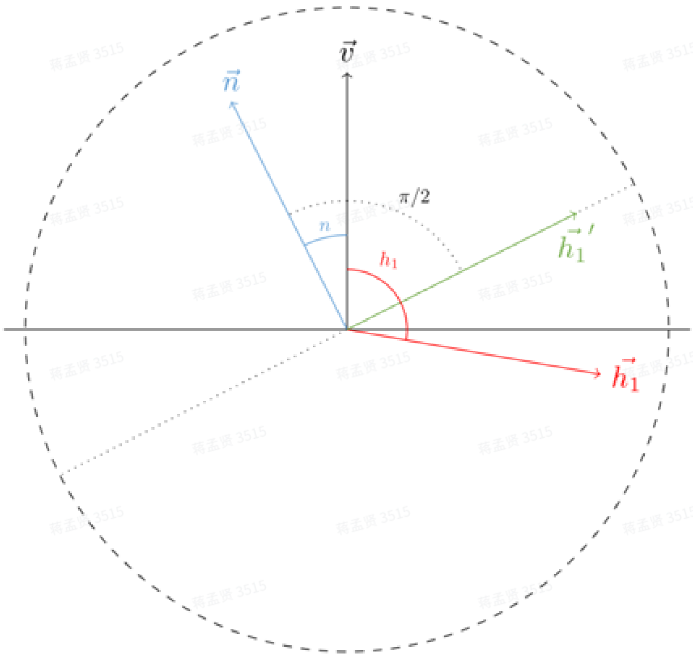
e. 计算 $(\vec{h}_1, \vec{h}_2)$ 和视线向量 $\vec{v}$ 的余弦值，并据此进行更新

```
// UE4 代码
SceneDepths.x = ConvertFromDeviceZ(LookupDeviceZ(UV2.xy));
SceneDepths.y = ConvertFromDeviceZ(LookupDeviceZ(UV2.zw));

V = ScreenToViewPos(UV2.xy, SceneDepths.x) - ViewPos;
LenSq = dot(V,V);
OOLen = rsqrt(LenSq + 0.0001);
Ang = dot(V,ViewDir) * OOLen;

FallOff = saturate(LenSq * AttenFactor);
Ang = lerp(Ang, BestAng.x, FallOff);
BestAng.x  = ( Ang > BestAng.x ) ? Ang : lerp( Ang, BestAng.x, Thickness );

// Negative Direction
V = ScreenToViewPos(UV2.zw, SceneDepths.y) - ViewPos;
LenSq = dot(V,V);
OOLen = rsqrt(LenSq + 0.0001);
Ang = dot(V,ViewDir) * OOLen;

```

```
FallOff = saturate(LenSq * AttenFactor);
Ang = lerp(Ang, BestAng.y, FallOff);

BestAng.y  = ( Ang > BestAng.y ) ? Ang : lerp( Ang, BestAng.y, Thickness );
```

Note: h_1 is negative

$$
\begin{aligned}
{h_1}' &= n + \max(h_1 - n, -\pi/2) \\
{h_2}' &= n + \min(h_2 - n, \pi/2)
\end{aligned}
$$

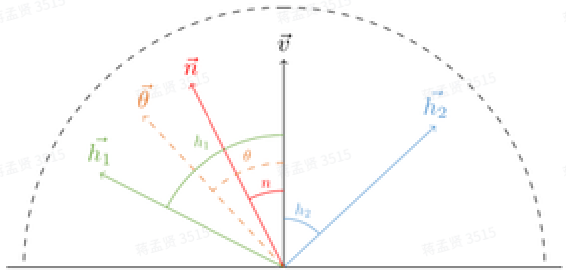


但 $(\vec{h}_1, \vec{h}_2)$ 在后面进行积分时，不能直接使用，因为半球采样域是以法线 $\vec{n}$ 为中心的，因此需要进行 `clamp` 操作，保证 $\vec{h}$ 和 $\vec{n}$ 之间的夹角不会超过 $\dfrac{\pi}{2}$。`Clamp` 方法如上所示。

> ✍️ 关于 $\vec{n}$ 的计算：见后文。注意：这里的法线不是View Space下的法线，而是还要进行一次切面投影！

继续推导

Cosine Weighting



- Ambient occlusion equation:

$$V_d^{cosine} = \frac{1}{\pi}\int_0^{\pi} \underbrace{\int_{-\pi/2}^{\pi/2} V(\theta,\phi)\cos(\theta-n)\,|\sin(\theta)|\,d\theta}_{v_d}\,d\phi$$

- Using horizon angles h_1 and h_2, we have the visibility for a single slice v_d:

$$v_d = IntegrateArc(h_1, h_2, n) =$$

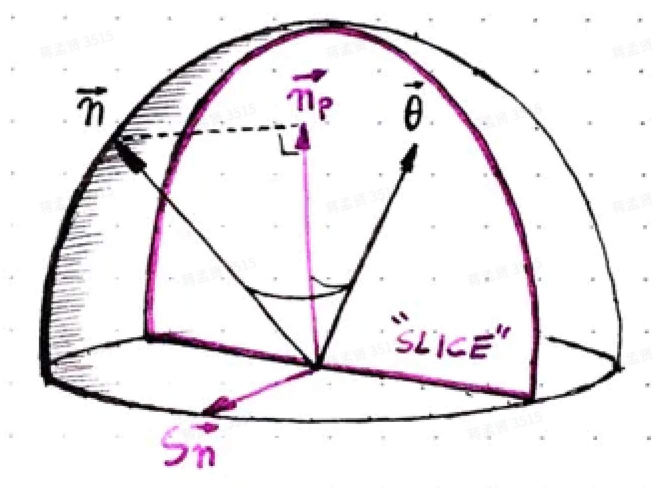
$$\int_0^{h_1}\cos(\theta-n)|\sin(\theta)|\,d\theta + \int_0^{h_2}\cos(\theta-n)|\sin(\theta)|\,d\theta =$$

$$\frac{1}{4}(-\cos(2h_1-n)+\cos(n)+2h_1\sin(n)) + \frac{1}{4}(-\cos(2h_2-n)+\cos(n)+2h_2\sin(n))$$

所以，对于每一个 ϕ 对应的切面积分，我们只需要计算上诉最终的公式，其中只有三个变量：$(\vec{h}_1, \vec{h}_2)$ 已知，而唯一只剩下 $\vec{n}$ 的计算。

计算切面投影法线

我们现在拥有哪些向量数据？思考下：View Space下的法线 $\vec{n}_{vs}$ ，视线向量 v ，采样向量 w_i （虽然是二维向量，但实际上可以看作View Space下的三维向量 $(\vec{w}_i, 0)$ ）。那么根据下图，我们可以有如下计算方法：（符号对不上，但是差不多是这个意思）




```
float3 PlaneNormal = normalize(cross(float3(ScreenDir.xy,0) ,ViewDir));
float3 ProjNormal = ViewSpaceNormal - PlaneNormal * dot(ViewSpaceNormal, PlaneNo
```

最终计算

将三个变量带入公式

2. UE4原代码

```
// 寻找最大垂直角对
float2 SearchForLargestAngleDual(uint NumSteps, float2 BaseUV, float2 ScreenDir,
{
    float SceneDepth, LenSq, OOLen, Ang, FallOff;
    float3 V;
    float2 SceneDepths =0;

    float2 BestAng = float2(-1,-1);
    float Thickness = GTAOParams[1].y;

    for(uint i=0; i<NumSteps; i++)
    {
        float fi = (float) i;

        float2 UVOffset = ScreenDir * max( SearchRadius * (fi + InitialOffset),
        UVOffset.y *= -1;
        float4 UV2 = BaseUV.xyxy + float4( UVOffset.xy, -UVOffset.xy );

// Positive Direction
        SceneDepths.x = ConvertFromDeviceZ(LookupDeviceZ(UV2.xy));
        SceneDepths.y = ConvertFromDeviceZ(LookupDeviceZ(UV2.zw));

        V = ScreenToViewPos(UV2.xy, SceneDepths.x) - ViewPos;
        LenSq = dot(V,V);
        OOLen = rsqrt(LenSq + 0.0001);
        Ang = dot(V,ViewDir) * OOLen;

        FallOff = saturate(LenSq * AttenFactor);
        Ang = lerp(Ang, BestAng.x, FallOff);
        BestAng.x  = ( Ang > BestAng.x ) ? Ang : lerp( Ang, BestAng.x, Thickness

// Negative Direction
        SceneDepths.x = ConvertFromDeviceZ(LookupDeviceZ(UV2.xy));
        SceneDepths.y = ConvertFromDeviceZ(LookupDeviceZ(UV2.zw));

        V = ScreenToViewPos(UV2.xy, SceneDepths.x) - ViewPos;
```

```
        LenSq = dot(V,V);
        OOLen = rsqrt(LenSq + 0.0001);
        Ang = dot(V,ViewDir) * OOLen;

        FallOff = saturate(LenSq * AttenFactor);
        Ang = lerp(Ang, BestAng.x, FallOff);
        BestAng.x  = ( Ang > BestAng.x ) ? Ang : lerp( Ang, BestAng.x, Thickness

        // Negative Direction
        V = ScreenToViewPos(UV2.zw, SceneDepths.y) - ViewPos;
        LenSq = dot(V,V);
        OOLen = rsqrt(LenSq + 0.0001);
        Ang = dot(V,ViewDir) * OOLen;

        FallOff = saturate(LenSq * AttenFactor);
        Ang = lerp(Ang, BestAng.y, FallOff);

        BestAng.y = ( Ang > BestAng.y ) ? Ang : lerp( Ang, BestAng.y, Thickness
    }

    BestAng.x = acosFast(clamp(BestAng.x, -1.0,  1.0));
    BestAng.y = acosFast(clamp(BestAng.y, -1.0,  1.0));

    return BestAng;
}



// 计算切面积分
// UV : UV坐标
// Angles : 成对最大角 h1 h2
// ScreenDir : 采样向量 Vec2
// ViewDir : 视图空间下的视线向量
// ViewSpaceNormal : 视图空间下的法线向量
// SceneDepth : 没用到?
float ComputeInnerIntegral(float2 UV, float2 Angles, float2 ScreenDir
, float3 ViewDir, float3 ViewSpaceNormal, float SceneDepth)
{
    // Given the angles found in the search plane we need to project the View Sp
    // 计算切面投影法线
    float3 PlaneNormal = normalize(cross(float3(ScreenDir.xy,0) ,ViewDir));
    float3 ProjNormal = ViewSpaceNormal - PlaneNormal * dot(ViewSpaceNormal, Pla

    // 和ViewDir, PlaneNormal构成坐标系的第三极
    // 功能上: 给定参考系, 确定角度
    float3 Perp = cross(ViewDir, PlaneNormal);

```

```
    // 计算法线的长度和其倒数，用于归一化
    float LenProjNormal                = length(ProjNormal) + 0.000001f;
    float RecipMag                       = 1.0f / (LenProjNormal);

    // sin(n)
    float CosAng                        = dot(ProjNormal, Perp) * RecipMag;
    // 角度: n
    float Gamma                                = acosFast(CosAng) - PI_HALF;
    // cos(n)
    float CosGamma                       = dot(ProjNormal, ViewDir) * RecipMag;
    // 2sin(n)
    float SinGamma                 = CosAng * -2.0f;

    // clamp to normal hemisphere
    // 范围合理化 Clamp
    Angles.x = Gamma + max(-Angles.x - Gamma, -(PI_HALF) );
    Angles.y = Gamma + min( Angles.y - Gamma,  (PI_HALF) );

    // 最终的积分公式
    float AO = ( (LenProjNormal) *  0.25 *
                                  ( (Angles.x * SinGamma + CosGamma - cos(
                                      (Angles.y * SinGamma + CosGamma - co

    return AO;
}
```