Ray Marching

1 Unity HDRP 的 Ray Marching

源代码在 RayMarching.hlsl 中，写该文档时对应的 HDRP 版本号为 14.0.12.

1.1 世界空间变换到屏幕空间

将世界空间下的坐标信息和前进方向转化为屏幕空间下的坐标信息和前进方向，确认射线能到达的边界（此处用 while 循环处理，设置最大步长）

```
// Extend and clip the end point to the frustum.
float tMax;
{
    // Shrink the frustum by half a texel for efficiency reasons.
    const float halfTexel = 0.5;

    float3 bounds;
    bounds.x = (rcpRayDir.x >= 0) ? _ScreenSize.x - halfTexel : halfTexel;
    bounds.y = (rcpRayDir.y >= 0) ? _ScreenSize.y - halfTexel : halfTexel;
    // If we do not want to intersect the skybox
    // it is more efficient to not trace too far.
    float maxDepth = (_RayMarchingReflectsSky != 0) ?
        -0.00000024 : 0.00000024; // 2^-22
    bounds.z = (rcpRayDir.z >= 0) ? 1 : maxDepth;

    float3 dist = bounds * rcpRayDir - (rayOrigin * rcpRayDir);
    tMax = Min3(dist.x, dist.y, dist.z);
}
```

1.2 射线前进

每次得到新的光线前进点，做一个 bias，避免光线位置落在边缘，做法如下：

```
// Ray position often ends up on the edge. To determine (and look up) the right cell,
// we need to bias the position by a small epsilon in the direction of the ray.
float2 sgnEdgeDist = round(rayPos.xy) - rayPos.xy;
float2 satEdgeDist = clamp(raySign.xy * sgnEdgeDist + RAY_TRACE_EPS, 0, RAY_TRACE_EPS);
rayPos.xy += raySign.xy * satEdgeDist;
```

1.3 使用 HiZ 去加速 Marching 的速度

1.3.1 计算 cube

计算判断 hit 的 cube，只考虑四个面

（a）其中两个面为光线的前进方向，比如光线在屏幕上朝右上方前进，则考虑 cube 的上面和右面；光线在屏幕上朝右下方前进，则考虑 cube 的右面和下面。

（b）固定考虑 cube 的前面和后面，后面对应 HiZ 采样的结果（深度较大值），前面对应靠近相机侧，可定义一个厚度来设置，具体方式见代码

```
// mipLevel is the level of the HiZ pyramid we want to sample.
int2 mipCoord  = (int2)rayPos.xy >> mipLevel;
// Bounds define 4 faces of a cube:
// 2 walls in front of the ray, and a floor and a base below it.
float4 bounds;

bounds.z  = LOAD_TEXTURE2D_X(_DepthTexture, mipOffset + mipCoord).r;
bounds.xy = (mipCoord + rayStep) << mipLevel;

// We define the depth of the base as the depth value as:
// b = DeviceDepth((1 + thickness) * LinearDepth(d))
// b = ((f - n) * d + n * (1 - (1 + thickness))) / ((f - n) * (1 + thickness))
// b = ((f - n) * d - n * thickness) / ((f - n) * (1 + thickness))
// b = d / (1 + thickness) - n / (f - n) * (thickness / (1 + thickness))
// b = d * k_s + k_b
bounds.w = bounds.z * _RayMarchingThicknessScale + _RayMarchingThicknessBias;
```

1.3.2 判定碰撞

注：英文为源代码注释，添加的中文为个人的理解

```
float4 dist      = bounds * rcpRayDir.xyzz - (rayOrigin.xyzz * rcpRayDir.xyzz);
float  distWall  = min(dist.x, dist.y);
float  distFloor = dist.z;
float  distBase  = dist.w;

// Note: 'rayPos' given by 't' can correspond to one of several depth values:
// - above or exactly on the floor
// - inside the floor (between the floor and the base)
// - below the base
bool belowFloor  = rayPos.z  < bounds.z;
bool aboveBase   = rayPos.z >= bounds.w;
// insideFloor用于判断rayPos是否在cube内且没有hit到后面（floor）
bool insideFloor = belowFloor && aboveBase;
// distFloor对应沿着光线前进会hit到cube的't'
// 当前的t必须小于等于才会hit到
// distFloor <= distWall表示光线会在cube内部hit到后面
// 也就是说光线会在hit到另外两个面之前，先hit到后面，这才算有效hit到后面
bool hitFloor    = (t <= distFloor) && (distFloor <= distWall);

// Game rules:
// * if the closest intersection is with the wall of the cell, switch to the coarser MIP,
// * if the closest intersection is with the heightmap below,  switch to the finer
MIP, and advance the ray.
// * if the closest intersection is with the heightmap above,  switch to the finer
MIP, and do NOT advance the ray.
// Victory conditions:
// * See below. Do NOT reorder the statements!

// 上一次步进没命中，且这一次也还是在cube内部，即表示为miss
miss = belowMip0 && insideFloor;
// 只有mipLevel=0，而且rayPos在cube内部或者hit到后面，则认为hit成功
hit = (mipLevel == 0) && (hitFloor || insideFloor);
// 目前已经是mipLevel=0，而且rayPos在cube内部，记录本次状态
belowMip0 = (mipLevel == 0) && belowFloor;

// 'distFloor' can be smaller than the current distance 't'.
```

```
35 // We can also safely ignore 'distBase'.
36 // If we hit the floor, it's always safe to jump there.
37 // If we are at (mipLevel != 0) and we are below the floor, we should not move.
38 t = hitFloor ? distFloor : (((mipLevel != 0) && belowFloor) ? t : distWall);
39 rayPos.z = bounds.z; // Retain the depth of the potential intersection
40
41 // Warning: both rays towards the eye, and tracing behind objects has linear
42 // rather than logarithmic complexity! This is due to the fact that we only store
43 // the maximum value of depth, and not the min-max.
44 mipLevel += (hitFloor || belowFloor || rayTowardsEye) ? -1 : 1;
45 mipLevel  = clamp(mipLevel, 0, 6);
46 mipOffset = _DepthPyramidMipLevelOffsets[mipLevel];
47 // mipLevel = 0;
```