SSPR反射

背景

传统的反射方案有：

1. 屏幕空间反射，利用深度图在屏幕空间反推世界空间来计算反射。特点：实时动态反射，开销稳定但计算量大。
2. 平面反射，使用一个对称摄像头再绘制一遍反射物体。特点：实时动态反射，开销不大但会随着场景复杂度提升，只能用于平面。
3. 反射cube，烘培实时或静态cube，通过采样cube得到反射。特点：实时cube开销大。
4. 屏幕空间平面反射：在屏幕空间下做平面翻转，开销小而稳定，没有平面反射随场景复杂度提升开销的特点。

目前游戏使用

```
1. 屏幕空间平面反射应对复杂度高、清晰度要求低的场景。
2. 平面反射应对复杂度低、清晰度高的场景。
```

不足：

```
1. 1024x1024的性能开销过大，只能选择512x512或256x256的分辨率，清晰度不足。
```

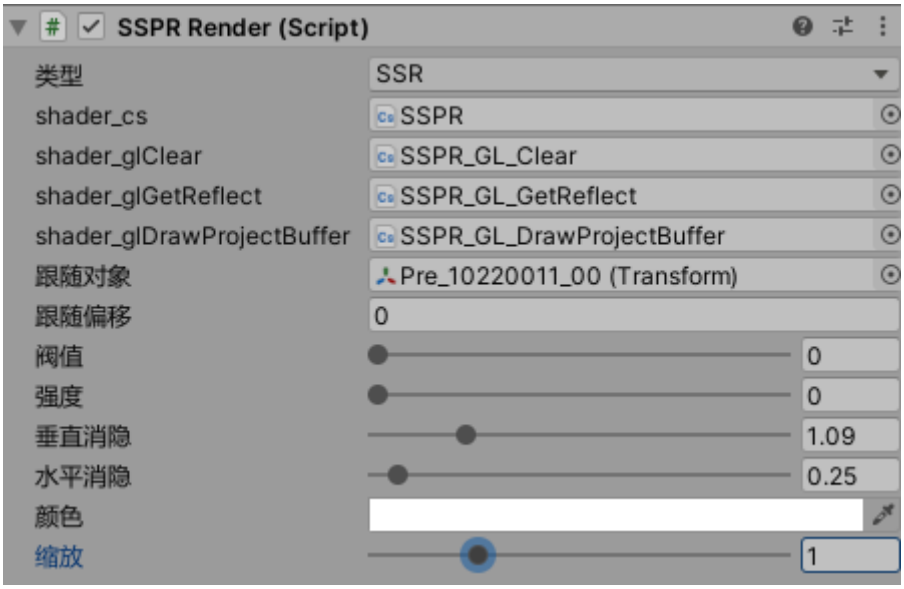
效果展示

操作指南：

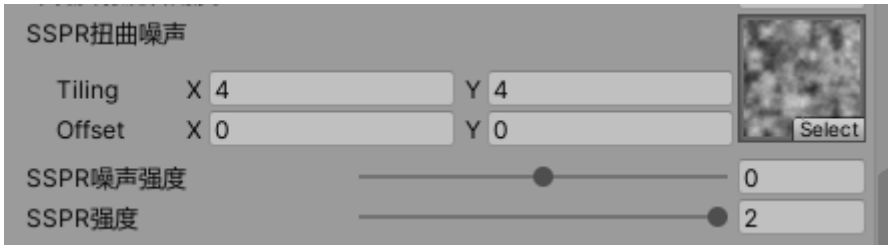
1. 在SceneRoot->Reflections->SSPRRender挂上SSPRRender脚本。
2. 地板选择SSPR_Bump脚本。
3. 单个窗口下运行调整参数。(scene和game视图不要同时开，现在运行才能看到)

参数说明：

1. SSPRRender:



    1. FollowTarget：跟随角色，控制反射的水平面。
    2. FollowOffset：水平面偏移，当平面偏下时，可能发生渲染错误。可以提高一点来避免。
    3. Screen LR Stretch Threshold：屏幕左右拉伸填充空位的阈值。(屏幕空间平面反射对左右两侧会有无法填充满的状况)
    4. Screen LR Stretch Intensity：屏幕左右拉伸填充空位的强度。
    5. FadeOutScreenBorderWidthVerticle：竖直方向渐隐
    6. FadeOutScreenBorderWidthHorizontal：水平方向渐隐
    7. FinalColor：混上某种颜色
    8. RTScale：RT精度，越往右精度越低（闪烁严重），性能越好。一倍缩放性能太糟糕，请从选择2或4倍
2. SSPR_Bump:

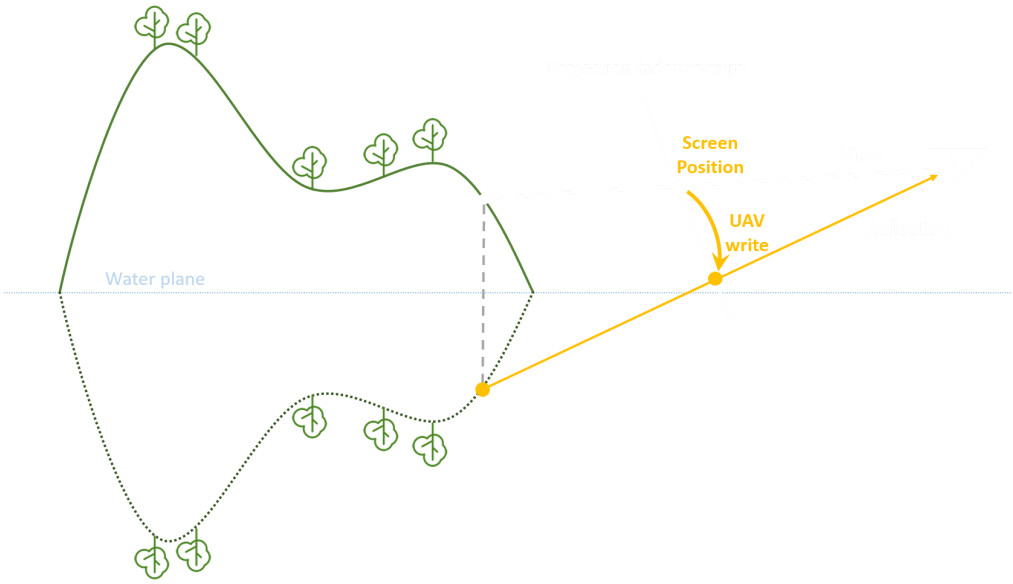


    1. SSPR扭曲噪声：使用噪声图形进行扭曲
    2. 噪声强度：扭曲强度
    3. SSPR强度：sspr反射强度

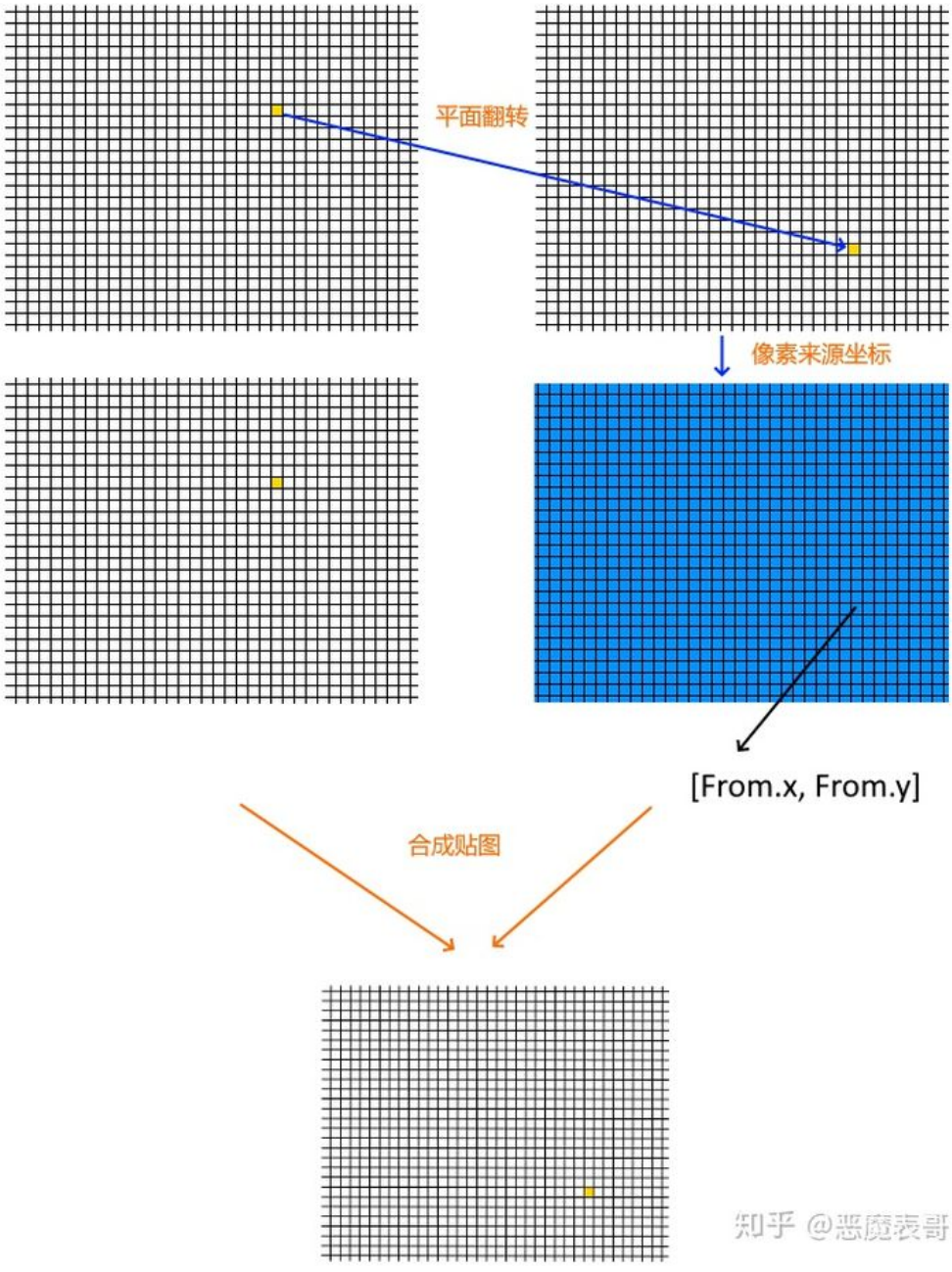
实现技术简介

1. 实现原理：在屏幕空间重建世界坐标，沿水平面翻转。再通过乱序写入(需要computer shader实现)，写入坐标编码(为了解决重叠的问题)，最后解码读原始图像得到屏幕空间屏幕反射图。
    1. 实现示意图：

1. 场景示意图：



2. compute shader 流程示意图：



3. 文字描述

    1. Clear Pass

1. 将编码图(OutputProjection)clear为最大编码值，因为使用了InterlockedMin排序。
2. 将反射结果图(Result)Clear为(0,0,0,0)

2. DrawProjectBuffer Pass 计算编码图，并写入
    1. 将屏幕坐标转成世界坐标posWS
    2. 沿反射平面翻转，得到镜像坐标reflectedPosWS
    3. 镜像坐标转成镜像屏幕坐标和镜像屏幕UV
    4. 依据镜像屏幕UV是否超出(0~1)判断是否超出屏幕范围。是，则结束计算。否，则继续。
    5. 依据镜像屏幕坐标，进行编码，写入OutputProjection
        1. 四个一组填补间隙(因为依据透视关系，镜像过来存在多对一和零对一的状况)
        2. 编码为Y:12，X:12,fade:8的方式，Y轴放在最前面是多对一时，取Y轴最小的像素(Y越小越靠前)。这里通过InterlockedMin得到多镜像中的最小值，因为各个平台对InterlockedMin不同。metal平台使用RWBuffer代替RWTexture2D
        3. fade是为了处理水平和数值方向的渐隐
3. GetReflect Pass 对编码图解码，写入反射结果图

2. 参考实现：
    1. UE4 屏幕空间平面反射源码，PostProcessPixelProjectedReflectionMobile.usf
    2. Unity URP 移动平台的屏幕空间平面反射（SSPR）趟坑记
    3. Screen Space Planar Reflections in Ghost Recon Wildlands
3. 关键源代码：

```
[numthreads(8,8,1)]
void DrawProjectBuffer(uint3 id : SV_DispatchThreadID)
{

//////////////////////////////////////////////////////////////////////////
////////////////////////////
    //Clear (clear挪出去，非URP使用会有问题)

//////////////////////////////////////////////////////////////////////////
////////////////////////////
    // uint3 tmpId = id;
    // Result[uint2(tmpId.xy)] = half4(0,0,0,0);//black rgb and alpha =
0. alpha 0 means no valid SSPR pixels found, so reflection plane will
not use SSRP's result
    // PosWSyRT[uint2(tmpId.xy)] = 9999999;//a very high posWS.y as
clear value
    uint2 ReflectedPixel = id.xy;

//////////////////////////////////////////////////////////////////////////
////////////////////////////
    //ConvertScreenIDToPosWS 将屏幕坐标转成世界坐标posWS

//////////////////////////////////////////////////////////////////////////
////////////////////////////
    float3 posWS = ConvertScreenIDToPosWS(ReflectedPixel.xy);
    float distance = length(_WorldSpaceCameraPos.xz-posWS.xz);
```

```
//////////////////////////////////////////////////////////////////////////
////////////////////////////
    //if posWS is already under reflection plane (e.g. under water
plane),
    //it will never be a correct color to reflect anyway, early exit to
prevent wrong result write to Color RT

//////////////////////////////////////////////////////////////////////////
////////////////////////////
    // if(posWS.y <= _PlanarY)
    //  return;


//////////////////////////////////////////////////////////////////////////
////////////////////////////
    //mirror posWS according to horizontal reflection plane (e.g. water
plane)
    //  沿反射平面翻转，得到镜像坐标

 //////////////////////////////////////////////////////////////////////////
////////////////////////////
    float3 reflectedPosWS = MirrorPosWS(posWS);


//////////////////////////////////////////////////////////////////////////
////////////////////////////
    //ConvertReflectedPosWSToScreenID 镜像坐标转成镜像屏幕坐标

//////////////////////////////////////////////////////////////////////////
////////////////////////////
    float2 reflectedScreenUV =
ConvertReflectedPosWSToScreenUV(reflectedPosWS);
    // 依据镜像屏幕UV是否超出(0~1)判断是否超出屏幕范围。是，则结束计算。否，则继续。
    //early exit if not valid uv anymore, to avoid out of bound access
    float2 earlyExitTest = abs(reflectedScreenUV - 0.5);
    if (earlyExitTest.x >= 0.5 || earlyExitTest.y >= 0.5)
        return;
    // 镜像屏幕坐标转成镜像屏幕UV
    float2 reflectedScreenID = (reflectedScreenUV * _Screen);//from
screen uv[0,1] to [0,RTSize-1]

    // 依据镜像屏幕坐标，进行编码，写入OutputProjection
    if(posWS.y-_PlanarY>0.01)
    {
        // 四个一组填补间隙(因为依据透视关系，镜像过来存在多对一和零对一的状况)
        for (int y = 0; y < 2; ++y)
        {
            for (int x = 0; x < 2; ++x)
            {
            int2 ReflectingPixel = floor(reflectedScreenID + half2(x,
y));
            ReflectingPixel.x = min(ReflectingPixel.x,
int(_Screen.x)-1);
            ReflectingPixel.y = min(ReflectingPixel.y,
int(_Screen.y)-1);
            float2 screenUV = ReflectedPixel.xy / _Screen;
            //fade是为了处理水平和数值方向的渐隐
```

```
            half fadeoutAlpha =
ConvertOpaqueColorRTScreenUVToFadeAlphaParam(screenUV, reflectedPosWS.y,
distance);
            uint fadeoutAlphaInt = fadeoutAlpha * 255;//8 bit
            // 编码为Y:12，X:12,fade:8的方式，Y轴放在最前面是多对一时，取Y轴最小
的像素(Y越小越靠前)。这里通过InterlockedMin得到多镜像中的最小值，因为各个平台对
InterlockedMin不同。metal平台使用RWBuffer代替RWTexture2D
            uint hash = ReflectedPixel.y << 20 | ReflectedPixel.x << 8 |
fadeoutAlphaInt;
            ProjectionBufferWrite(ReflectingPixel,hash);
            }
        }
    }
    // else
    // {
    //  ProjectionBufferWrite(reflectedScreenID,PROJECTION_PLANE_VALUE);
    // }
}
```