B站：https://www.bilibili.com/video/BV1eV411r7PD/

以上是一个早期的URP版本，仍有许多问题没有解决。场景美术资源采用The Illustrated Nature，令人惊叹的充满爱的艺术品。当然以上都不会提供。本文的水面模型仅仅是Plane，无需美术加工。

本文要讲述的是HDRP下的风格化水体渲染，源码已内置于JTRP：

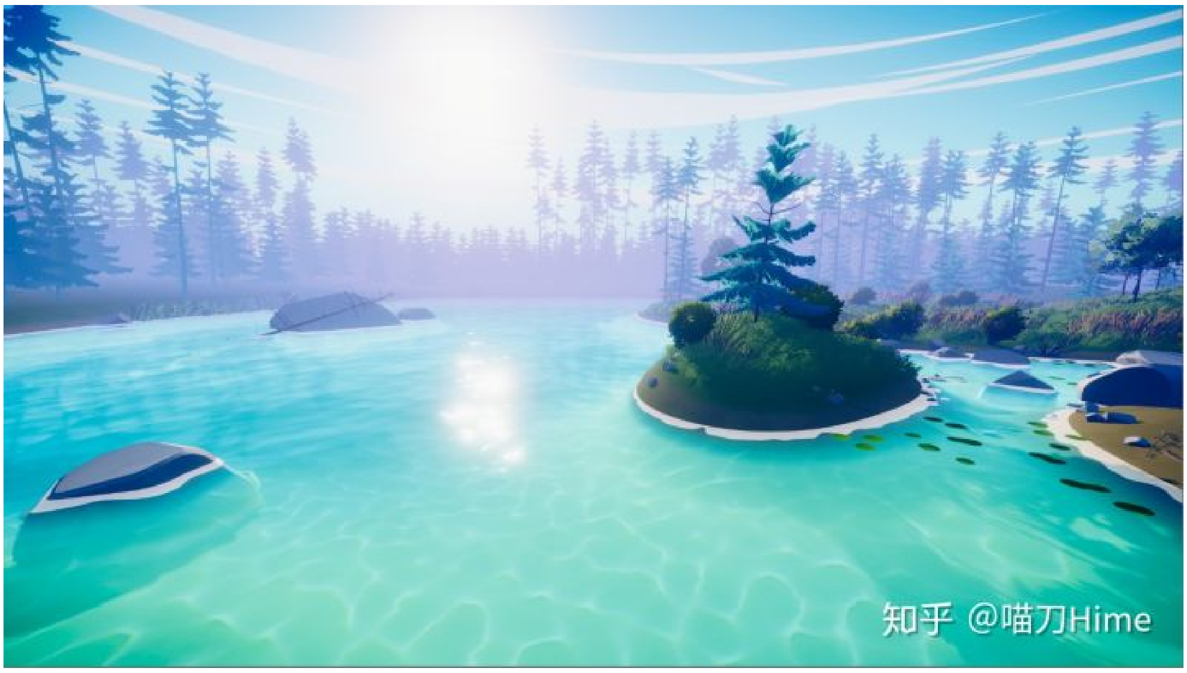
JasonMaToonRenderPipelinegithub.com

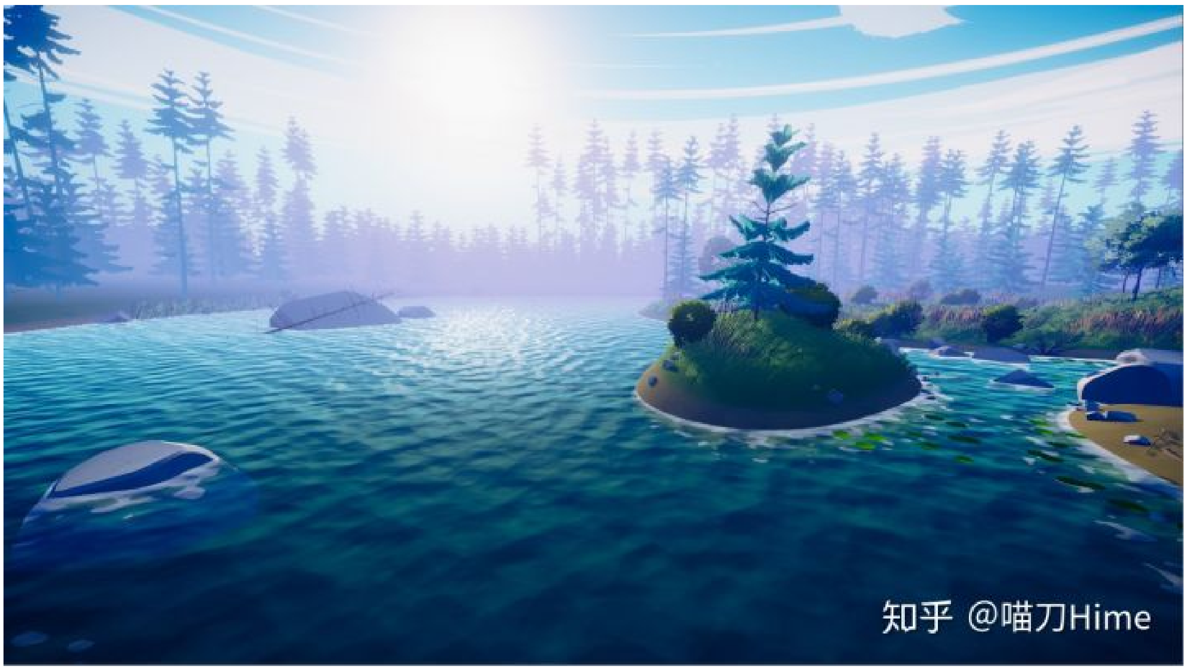
包含以下特性：

- Absorption & Scattering —— 使用Ramp模拟水的吸收和散射
- Refraction & Distortion —— 对屏幕UV施以扰动模拟折射扭曲
- Caustics —— 简单的贴图模拟焦散
- Specular —— 抄来的简化GGX高光
- Reflections & Fresnel —— 依旧是抄来的使用反射探针的简化反射
- Foam —— 最简单无脑的采样Perlin Noise的泡沫效果
- Tessellation —— 半天自学的曲面细分以配合微弱的顶点动画

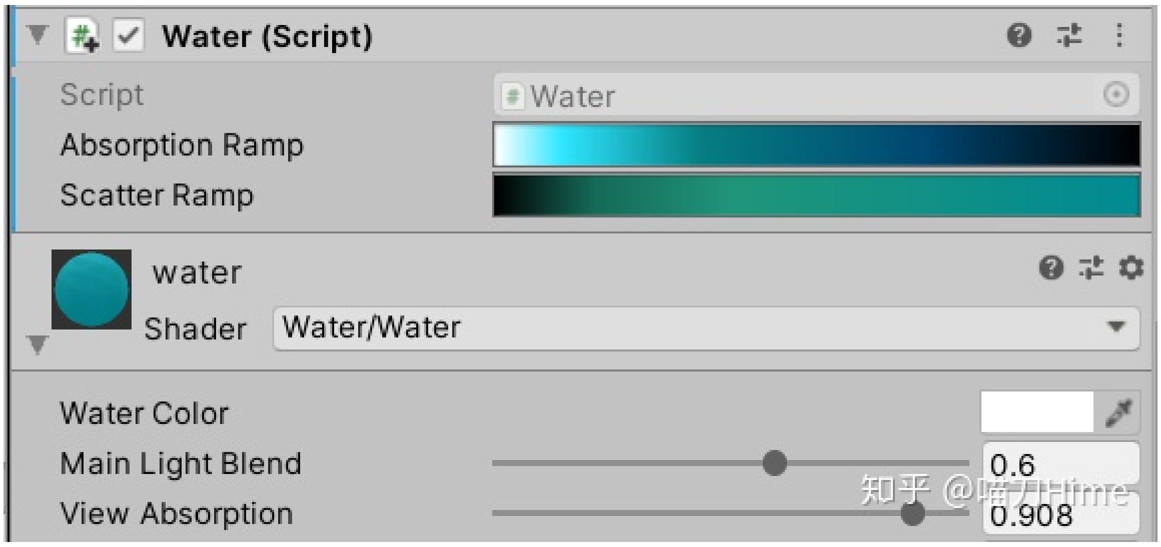
相对于早期URP版本多了Tessellation、改进的折射和一些算法，不过美术资源用不了了，只能拿默认场景将就着看了。关于近岸泡沫部分，本想用计算替代这篇文章中刷UV等美术过程，先后尝试了Compute Shader生成Mesh、2D SDF等多个方案，之后由于时间原因还是采用了最简单的方案。另外大量借鉴了官方URP示例Boat Attack，比如Tessellation和着色部分，而Boat Attack的海浪做的相当复杂，包含大量Jobs和物理运算，这里直接偷懒采样贴图完事。

然后再放下场景自带的氵 和本文的氵 的对比图：





Absorption & Scattering

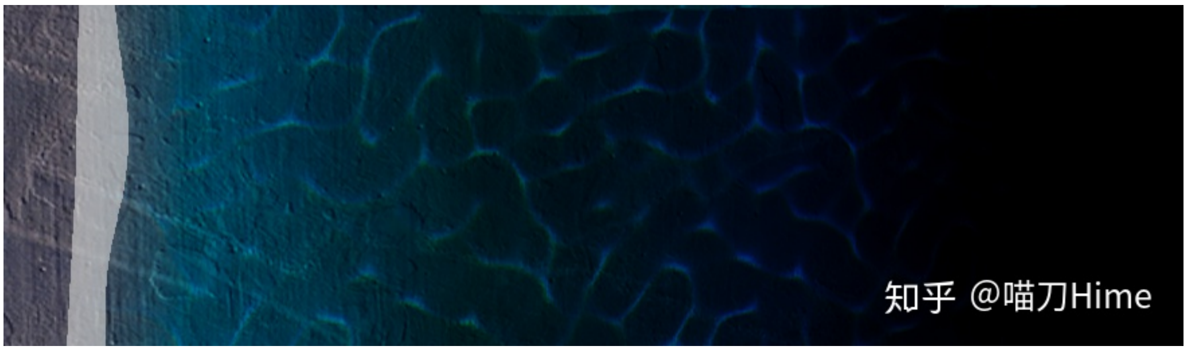


Water (Script)

Script: Water
Absorption Ramp
Scatter Ramp

water
Shader: Water/Water

Water Color
Main Light Blend: 0.6
View Absorption: 0.908



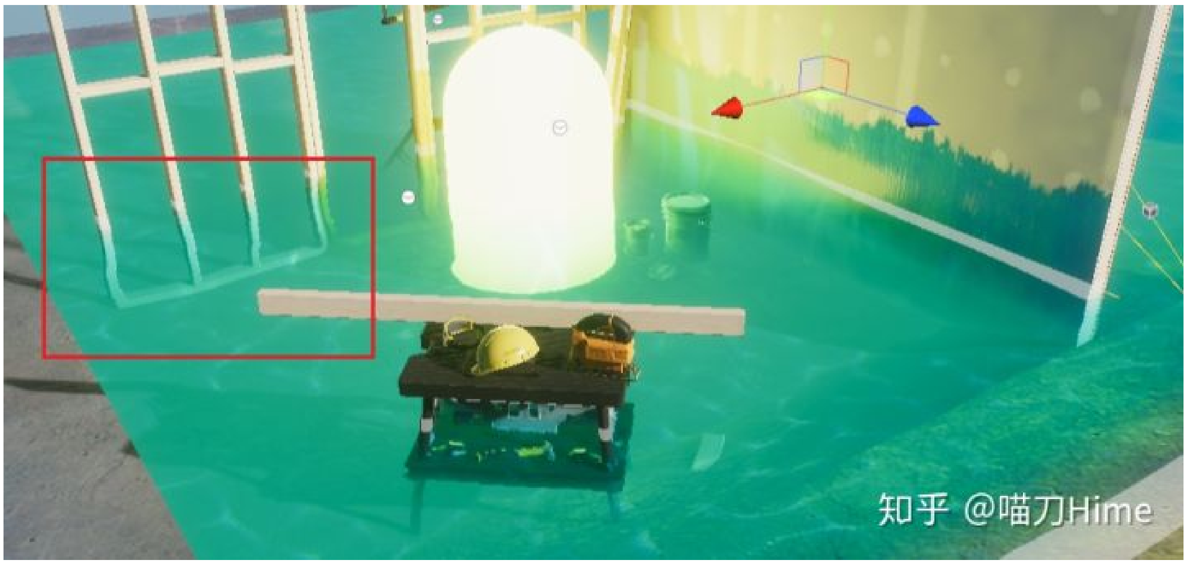
Absorption

Scattering

使用Ramp模拟水的吸收和散射，_ViewAbsorption控制水的**世界深度**（在世界坐标Y轴上的深度）和**视角深度**（摄像机坐标Z轴上的深度）的混合，世界深度固定不变，视角深度则会随摄像机视角而改变，混合两者则可以模拟一种“俯瞰水显得清澈，平视水显得深邃”的效果。

```
float absorptionDepth = depthWaterWorld * lerp(depthWater, 1, _ViewAbsorption) *
depthFactor;
finalColor.rgb += Scattering(absorptionDepth) * lighting * _waterColor;
finalColor.rgb += reflection;
finalColor.rgb += foam;
finalColor.rgb += (screenColor + caustics * lighting) *
Absorption(absorptionDepth);
```

Refraction & Distortion



光在水和空气间传播会产生折射，可以对屏幕空间UV施加扰动模拟这一现象。并且根据水深的不同折射程度也不同，还要注意只会折射水下的物体。实际采样颜色的同时还要采样深度。

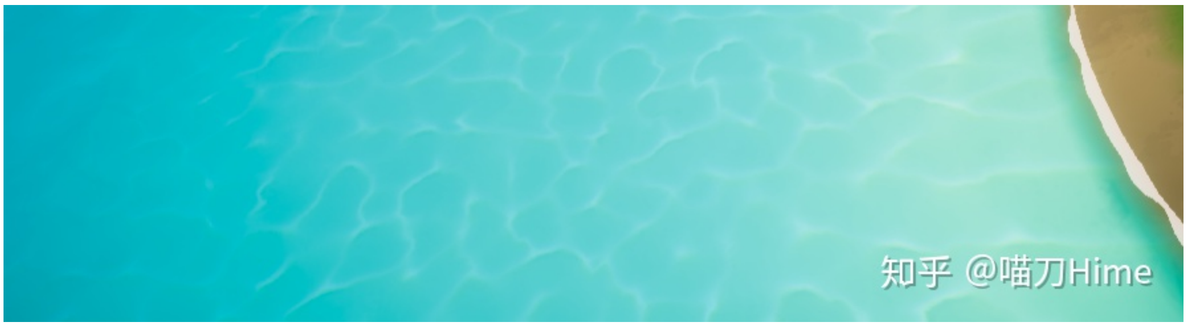
还有一点需要注意，如下图红框所示，折射在越接近水面的地方越弱，所以需要额外采样一次深度才能确定最终对屏幕UV的扰动幅度。

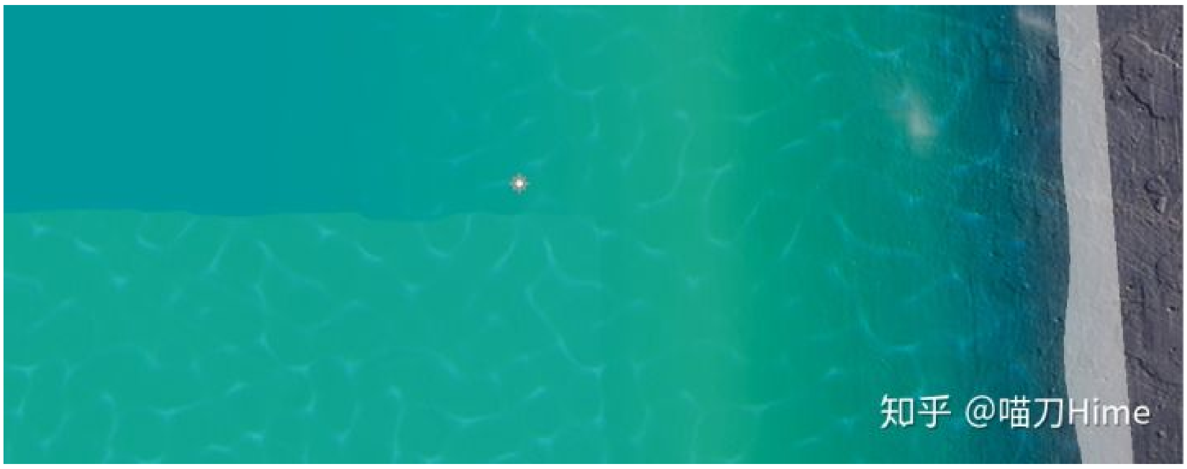
```
void GetDistortionSSData(inout float2 screenPos, float2 distortionUV, float3
viewPos, float3 worldPos, inout float depthTex,
inout float depthScene, inout float depthWater, inout float3 sceneWorldPos,
inout float depthWaterWorld, inout float3 screenColor)
{
    // 随距离缩放且防止边缘溢出
    distortionUV /= 1 + abs(viewPos.z);
    distortionUV *= saturate((1 - screenPos) * 20);
    float _depthTex = SampleCameraDepth(screenPos + distortionUV);
    float _depthScene = LinearEyeDepth(_depthTex, _ZBufferParams);
    float _depthWater = (_depthScene + viewPos.z);
    float3 _sceneWorldPos = ReconstructionWorldPos(screenPos + distortionUV,
_depthTex);
    float _depthWaterWorld = (worldPos.y - _sceneWorldPos.y);

    // 再次采样以随水深变浅减少折射
    distortionUV *= min(saturate(_depthWaterWorld), saturate(depthWaterWorld));
    _depthTex = SampleCameraDepth(screenPos + distortionUV);
    _depthScene = LinearEyeDepth(_depthTex, _ZBufferParams);
    _depthWater = (_depthScene + viewPos.z);
    _sceneWorldPos = ReconstructionWorldPos(screenPos + distortionUV,
_depthTex);
    _depthWaterWorld = (worldPos.y - _sceneWorldPos.y);

    // 仅在水盖住物体时折射
    if (_depthWater > 0)
    {
        depthTex = _depthTex;
        depthScene = _depthScene;
        depthWater = _depthWater;
        sceneWorldPos = _sceneWorldPos;
        depthWaterWorld = _depthWaterWorld;
        screenPos += distortionUV;
    }
    screenColor = SampleCameraColor(screenPos);
}
```

Caustics





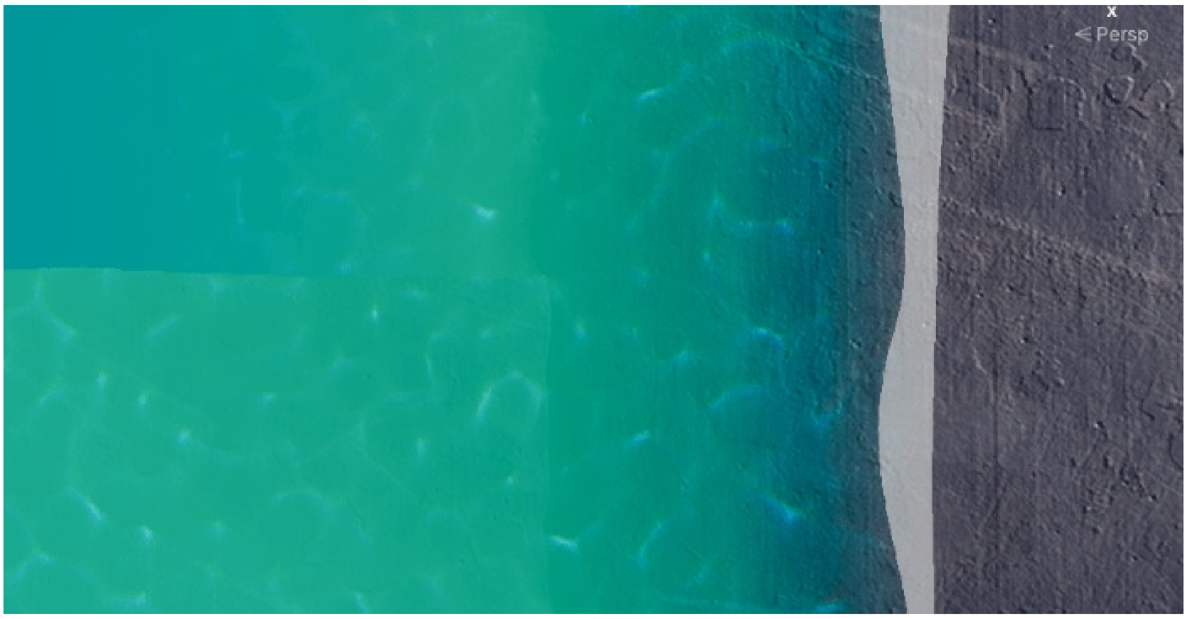
用纹理描述焦散效果，没啥好说的，也可以满足不同风格的视觉效果。对焦散纹理的采样则需要用深度图反推的水底世界坐标，如此才能保证纹理被均匀地映射到水底表面。另外和折射一样，焦散强度也会受深度影响，*depthBlendScale和*depthBlendPower控制了焦散和折射在浅水处的衰减程度。

而深度反推世界坐标在HDRP则麻烦一些，首先HDRP的世界坐标是相对于相机的，M2W之后世界坐标的原点在摄像机上，需要注意转换。另外HDRP的IVP矩阵的XZ轴由于不明原因旋转方向与Builtin和URP相反，坑了我好久。

```
// World Position reconstruction from Depth.
float3 ReconstructionWorldPos(float2 screenPos, float depthTex)
{
    // 不明原因使IVP矩阵的XZ轴旋转相反，这里进行矫正
    float4x4 IVP = UNITY_MATRIX_I_VP;
    IVP._12_32 *= -1;
    IVP[1].xyz *= -1;
    float4 sceneWorldPos = mul(IVP, float4((screenPos * 2 - 1), depthTex, 1));
    sceneWorldPos /= sceneWorldPos.a;// homogeneous coordinates
    return GetAbsolutePositionWS(sceneWorldPos.xyz);// 从Camera-relative Space还
原世界坐标
}

float3 GetCaustics(float2 t, float3 sceneWorldPos, float3 sceneNormal)
{
    // sceneNormal.y系数控制竖直方向采样速度，xz控制水平采样方向
    float2 uv = t + TRANSFORM_TEX((sceneWorldPos.xz + sceneWorldPos.y * -
(sceneNormal.x + sceneNormal.z) * (sceneNormal.y + 0.5) * 0.5), _MainTex);
    return SAMPLE_TEXTURE2D(_MainTex, s_linear_repeat_sampler, uv).rgb;// World
space texture sampling
}
```

Better Caustics



感谢Colin大佬的补充，额外增加一次反向的采样使焦散更加随机：

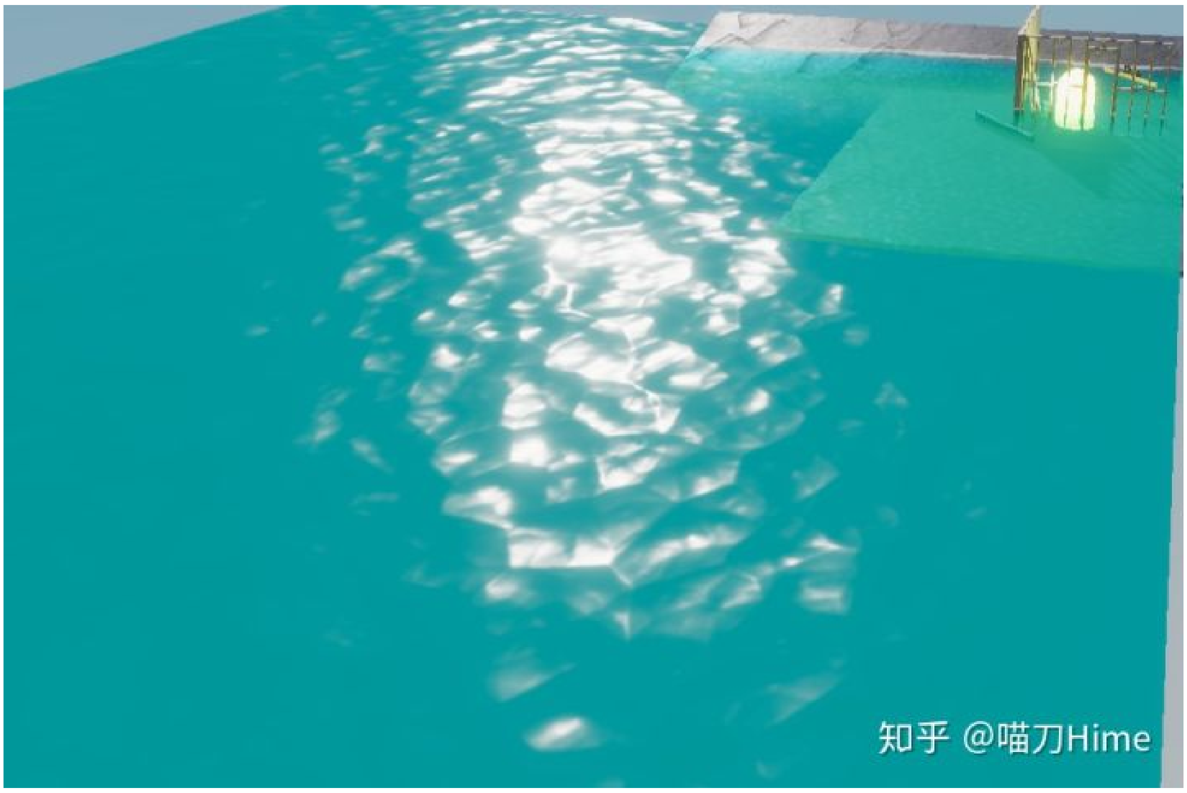
```
float3 GetCaustics(float2 t, float3 sceneWorldPos, float3 sceneNormal)
{
    // extract shared uv
    // sceneNormal.y系数控制竖直方向采样速度，xz控制水平采样方向
    float2 uv = sceneWorldPos.xz + sceneWorldPos.y * -(sceneNormal.x +
sceneNormal.z) * (sceneNormal.y + 0.5) * 0.5;

    // original caustics
    float2 uv1 = t + TRANSFORM_TEX(uv, _MainTex);
    float3 result1 = SAMPLE_TEXTURE2D(_MainTex, s_linear_repeat_sampler,
uv1).rgb;// World space texture sampling
    //return result1; //enable this line will return Jason's original caustics
result

    // sample caustics texture with an opposite flow direction uv again, the
goal is to make caustics more random, caustics will never flow to a single
direction uniformly
    float2 uv2 = t * float2(-1.07,-1.437) * 1.777 + TRANSFORM_TEX(uv,
_MainTex);//any opposite direction uv, as long as uv2 is not equals uv1, it
should looks good
    uv2 *= 0.777; //make texture bigger
    float3 result2 = SAMPLE_TEXTURE2D(_MainTex, s_linear_repeat_sampler,
uv2).rgb;// World space texture sampling

    float intensityFix = 4; //because we will use min() next line, overall
result will be darker, use a multiply to fix it
    return min(result1, result2) * intensityFix; //min() is the magic function
of rendering fastest fake caustics!
}
```

Specular



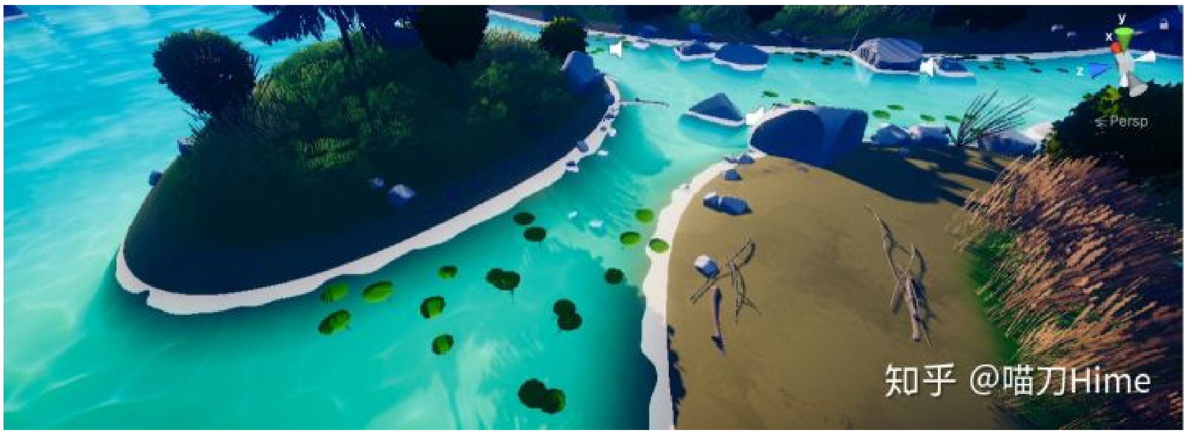
从BoatAttack照抄的简化版GGX，没啥好说的。

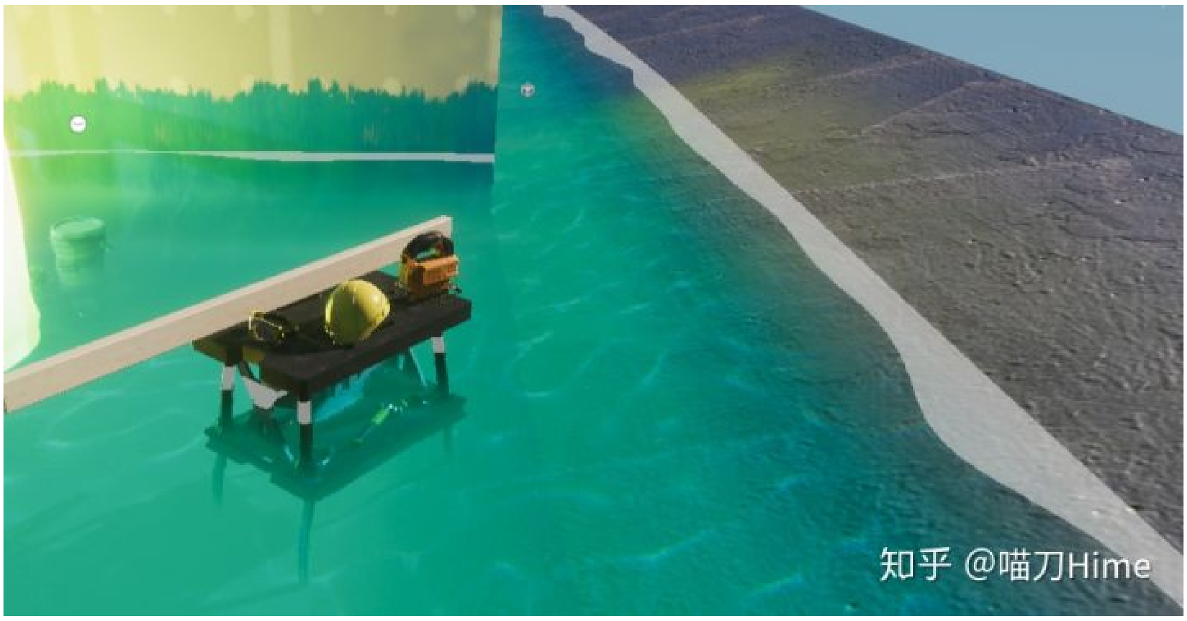
```
half3 Highlights(half3 positionWS, half roughness, half3 normalWS, half3
viewDirectionWS)
{
    Light mainLight = GetMainLight();
    half roughness2 = roughness * roughness;
    half3 halfDir = SafeNormalize(mainLight.direction + viewDirectionWS);
    half NoH = saturate(dot(normalize(normalWS), halfDir));
    half LoH = saturate(dot(mainLight.direction, halfDir));
    // GGX Distribution multiplied by combined approximation of Visibility and
Fresnel
    half d = NoH * NoH * (roughness2 - 1.h) + 1.0001h;
    half LoH2 = LoH * LoH;
    half specularTerm = roughness2 / ((d * d) * max(0.1h, LoH2) * (roughness +
0.5h) * 4);
    return specularTerm * mainLight.color * mainLight.distanceAttenuation;
}
```

Reflections & Fresnel

这里用的是CubeMap作为反射源，简单粗暴，不过完全不正确。也可以改成平面反射等其他更准确的方法。代码就不用放了。

Foam

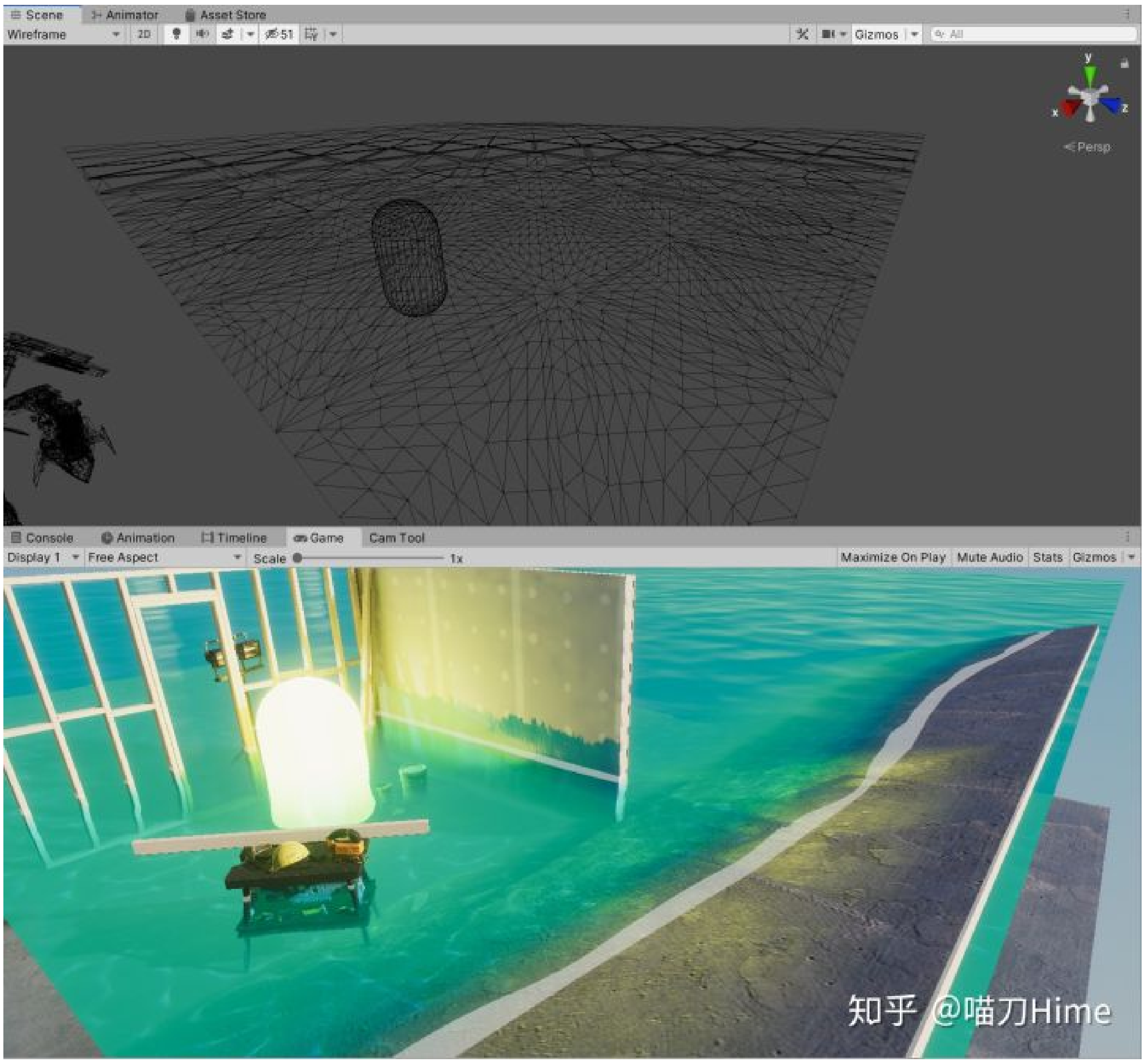




泡沫总体实现过程挺曲折的，最后还是用了最简单的方法，使用水体的世界深度作为阈值采样Perlin Noise。虽然没什么时间了，不过我觉得预计算2D SDF还是有可能替代此文的美术过程同时保持效果差不多的。

```
float3 Foam(float worldWaterDepth, float3 worldPos, float2 t)
{
    float foamRange = 1 - pow(saturate(worldWaterDepth * _FoamRange), 0.75);
    float foamNoise = SAMPLE_TEXTURE2D(_FoamNoise, s_linear_repeat_sampler, t + 
TRANSFORM_TEX((worldPos.xz), _FoamNoise)).r;
    foamNoise = PositivePow(foamNoise, _NoisePower);
    float3 foam = step(foamNoise, foamRange);
    foam *= _FoamColor.rgb * _FoamColor.a;
    return foam;
}
```

Tessellation



为了做顶点动画临时学的曲面细分，效果尚可但我觉得还有改进的地方，应该以屏幕空间包围盒面积作为Tessellation Factor，这样三角形应该会分布得更加均匀。

代码部分基本是从Boat Attack抄的，在Domain函数中做了简单的顶点动画，推荐Catlike Coding的Tessellation入门：

https://catlikecoding.com/unity/tutorials/advanced-rendering/tessellation/catlikecoding.com

```
struct TessellationControlPoint
{
    float4 vertex: INTERNALTESSPOS;
    float4 texcoord: TEXCOORD0; // Geometric UVs stored in xy, and world(pre-
waves) in zw
    float3 posWS: TEXCOORD1;    // world position of the vertices
    UNITY_VERTEX_INPUT_INSTANCE_ID
};

struct HS_ConstantOutput
{
    float TessFactor[3]: SV_TessFactor;
    float InsideTessFactor: SV_InsideTessFactor;
};


float TessellationEdgeFactor(float3 p0, float3 p1)
{
    float edgeLength = distance(p0, p1);

    float3 edgeCenter = (p0 + p1) * 0.5;
    float viewDistance = smoothstep(_TessellationStart, _TessellationEnd,
length(edgeCenter)) * 100 + 1;

    return edgeLength * _ScreenParams.y / (_TessellationEdgeLength *
viewDistance);
}

TessellationControlPoint TessellationVertex(VertexInput v)
{
    TessellationControlPoint o;
    o.vertex = v.vertex;
    o.posWS = TransformObjectToWorld(v.vertex.xyz);
    o.texcoord.xy = v.uv0.xy;
    o.texcoord.zw = o.posWS.xz;
    //o.color = v.color;
    return o;
}

HS_ConstantOutput HSConstant(InputPatch < TessellationControlPoint, 3 > Input)
{
    float3 p0 = TransformObjectToWorld(Input[0].vertex.xyz);
    float3 p1 = TransformObjectToWorld(Input[1].vertex.xyz);
    float3 p2 = TransformObjectToWorld(Input[2].vertex.xyz);
    HS_ConstantOutput o = (HS_ConstantOutput)0;
    o.TessFactor[0] = TessellationEdgeFactor(p1, p2);
    o.TessFactor[1] = TessellationEdgeFactor(p2, p0);
    o.TessFactor[2] = TessellationEdgeFactor(p0, p1);
    o.InsideTessFactor = (TessellationEdgeFactor(p1, p2) +
    TessellationEdgeFactor(p2, p0) +
    TessellationEdgeFactor(p0, p1)) * (1 / 3.0);
    return o;
}

[domain("tri")]
[partitioning("fractional_odd")]
[outputtopology("triangle_cw")]
[patchconstantfunc("HSConstant")]
```

```
[outputcontrolpoints(3)]
TessellationControlPoint Hull(InputPatch < TessellationControlPoint, 3 > Input,
uint uCPID: SV_OutputControlPointID)
{
    return Input[uCPID];
}

// Domain: replaces vert for tessellation version
[domain("tri")]
VertexOutput Domain(HS_ConstantOutput HSConstantData, const OutputPatch <
TessellationControlPoint, 3 > Input, float3 BarycentricCoords:
SV_DomainLocation)
{
    VertexOutput o = (VertexOutput)0;
    UNITY_SETUP_INSTANCE_ID(v);
    UNITY_TRANSFER_INSTANCE_ID(v, o);
    UNITY_INITIALIZE_VERTEX_OUTPUT_STEREO(o);
    /////////////////////Tessellation//////////////////////
    float fU = BarycentricCoords.x;
    float fV = BarycentricCoords.y;
    float fW = BarycentricCoords.z;

    float4 coords = Input[0].texcoord * fU + Input[1].texcoord * fV +
Input[2].texcoord * fW;
    o.uv0 = coords.xy;
    o.worldPos = Input[0].posWS * fU + Input[1].posWS * fV + Input[2].posWS *
fW;

    float2 worldUV = GetAbsolutePositionWS(o.worldPos).xz;
    o.worldPos.y += (SAMPLE_TEXTURE2D_LOD(_NormalMap, s_linear_repeat_sampler,
    worldUV * half2(0.0025, 0.0025) + _Time.x * 0.075, 0).r * 2 - 1) * 0.5;
    o.viewPos = mul(UNITY_MATRIX_V, float4(o.worldPos, 1)).xyz;
    o.clipPos = mul(UNITY_MATRIX_P, float4(o.viewPos, 1));
    o.screenPos = ComputeScreenPos(o.clipPos, _ProjectionParams.x);
    o.viewDir = GetWorldSpaceNormalizeViewDir(o.worldPos);
    return o;
}
```