

指数高度雾

背景

这里照搬了UE4的指数高度雾(不包含体积雾)。可以明显体现出雾效在地面堆积的感觉。其实现是模拟视线上雾浓度的积分，因此有一定的体积感。本来打算参照**filament**对其计算进行简化，但美术不想砍效果（例如**次要雾**、**定向内散射**），所以没有动。之后可以考虑简化。

效果展示



2. 使用说明:

1. 给任意节点挂上ExponentialHeightFogCtrl
2. 将Sun灯挂到脚本的方向光上
3. 调整参数
3. 参数说明: [指数高度雾用户指南](#)

指数高度雾组件	
雾密度 (Fog Density)	此为整体密度系数，是可视雾层的厚度。
雾高度衰减 (Fog Height Falloff)	高度密度系数，控制高度降低时密度增加的程度。值越小，过渡就越大。
高度雾偏移 (Height Fog Offset)	此控制相对于Actor放置Z（高度）的雾层高度偏移。
次要雾数据 (Second Fog Data)	此类设置控制次要雾层。将该次要雾层的 雾密度 设为 0 将不会产生影响。 雾密度 (Fog Density) ：次要雾层的整体密度系数，可用于添加另一雾层厚度。 雾高度衰减 (Fog Height Falloff) ：次要雾层的高度密度系数，控制高度降低时密度的增大程度。值越小，过渡就越大。 雾高度偏移 (Fog Height Offset) ：场景中相对于Actor Z高度位置的高度偏移。
雾内散射颜色 (Fog Inscattering Color)	设置雾的内散射颜色。此本质为雾的主要颜色。
雾最大不透明度 (Fog Max Opacity)	控制雾的最大不透明度。值为1时雾完全不透明，值为0时雾基本不可见。
开始距离 (Start Distance)	雾出现处与摄像机的距离。
雾衰减距离 (Fog Cutoff Distance)	雾不会被应用到超过此距离的场景元素。此属性有助于排除已烘焙雾的天空盒。
定向内散射	
定向内散射指数 (Directional Inscattering Exponent)	控制定向内散射椎体大小，用于估算定向光源的内散射。
定向内散射开始距离 (Directional Inscattering Start Distance)	控制定向内散射查看器的开始距离，用于估算定向光源的内散射。

指数高度雾组件	
定向内散射颜色 (Directional Inscattering Color)	设置定向内散射颜色，用于估算定向光源的内散射。此设置与定向光源模拟颜色的调整类似。

实现技术简介

1. 技术参考：

1. [高度雾ExponentialHeightFog](#):这篇文章简要介绍了ue4的技术实现。
2. [ig高度雾注解](#):这篇更贴近指数高度雾的技术原理，代码也更简洁，只是和ue4的实现离得更远。
3. [filament指数高度雾](#):filament项目中的指数高度雾，相比UE更简化。

2. 技术简介

1. 公式推导

$$r_y = O_y + t * k_y \text{ 视线方程的y分量为: } r_y = O_y + t * k_y$$

假设雾的浓度d随高度呈指数衰减，即： $d = a * e^{-b*y}$

则视线OP上的浓度积分为：

$$\begin{aligned}
 D &= \int_0^T d dt \\
 &= \int_0^T a * e^{-b*r_y} dt \\
 &= \int_0^T a * e^{-b*(O_y+t*k_y)} dt \\
 &= a * e^{-b*O_y} \int_0^T e^{-b*t*k_y} dt \\
 &= a * e^{-b*O_y} \int_0^{-b*T*k_y} e^{-b*t*k_y} \frac{1}{-b * k_y} d(-b * t * k_y) \\
 &= a * e^{-b*O_y} \frac{1}{-b * k_y} e \Big|_0^{-b*T*k_y} \\
 &= a * e^{-b*O_y} \frac{1}{-b * k_y} (e^{-b*T*k_y} - e^0) \\
 &= a * e^{-b*O_y} \frac{1 - e^{-b*T*k_y}}{b * k_y}
 \end{aligned}$$

opticalThickness = D

fogAmount=1-extinction= $1 - e^{-opticalThickness}$

2. 核心代码

```
// UE 4.22 HeightFogCommon.usf
// @param WorldPositionRelativeToCamera = WorldPosition - InCameraPosition
half4 GetExponentialHeightFog(float3 WorldPositionRelativeToCamera) // camera to
vertex
{
    // 雾效最高透明度
    const half MinFogOpacity = ExponentialFogColorParameter.w;

    const float3 worldObserverOrigin = float3(_worldSpaceCameraPos.x,
        min(_worldSpaceCameraPos.y, 65535), _worldSpaceCameraPos.z);
    // Receiver 指着色点
    float3 CameraToReceiver = WorldPositionRelativeToCamera;
    CameraToReceiver.y += _worldSpaceCameraPos.y - worldObserverOrigin.y;
    float CameraToReceiverLengthSqr = dot(CameraToReceiver,
        CameraToReceiver);
    float CameraToReceiverLengthInv = rsqrt(CameraToReceiverLengthSqr); //
    平方根的倒数
    float CameraToReceiverLength = CameraToReceiverLengthSqr *
        CameraToReceiverLengthInv;
    half3 CameraToReceiverNormalized = CameraToReceiver *
        CameraToReceiverLengthInv;

    // 雾效浓度
    // FogDensity * exp2(-FogHeightFalloff * (CameraWorldPosition.y -
    FogHeight))
    float RayOriginTerms = ExponentialFogParameters.x;
    float RayOriginTermsSecond = ExponentialFogParameters2.x;
    float RayLength = CameraToReceiverLength;
    float RayDirectionY = CameraToReceiver.y;

    // Factor in StartDistance
    // ExponentialFogParameters.w 是 StartDistance
    float ExcludeDistance = ExponentialFogParameters.w;
    // ExcludeDistance = 1.0;
    if (ExcludeDistance > 0)
    {
        // 到相交点所占时间
        float ExcludeIntersectionTime = ExcludeDistance *
            CameraToReceiverLengthInv;
        // 相机到相交点的 y 偏移
        float CameraToExclusionIntersectionY = ExcludeIntersectionTime *
            CameraToReceiver.y;
        // 相交点的 y 坐标
        float ExclusionIntersectionY = worldObserverOrigin.y +
            CameraToExclusionIntersectionY;
        // 相交点到着色点的 y 偏移
        float ExclusionIntersectionToReceiverY = CameraToReceiver.y -
            CameraToExclusionIntersectionY;

        // Calculate fog off of the ray starting from the exclusion
        distance, instead of starting from the camera
        // 相交点到着色点的距离
        RayLength = (1.0f - ExcludeIntersectionTime) *
            CameraToReceiverLength;
        // 相交点到着色点的 y 偏移
```

```

RayDirectionY = ExclusionIntersectionToReceiverY;
// ExponentialFogParameters.y : height falloff
// ExponentialFogParameters3.y : fog height
// height falloff * height
float Exponent = max(-127.0f, ExponentialFogParameters.y *
(ExclusionIntersectionY - ExponentialFogParameters3.y));
// ExponentialFogParameters3.x : fog density
RayOriginTerms = ExponentialFogParameters3.x * exp2(-Exponent);

// ExponentialFogParameters2.y : FogHeightFalloffSecond
// ExponentialFogParameters2.w : fog height second
float ExponentSecond = max(-127.0f, ExponentialFogParameters2.y *
(ExclusionIntersectionY - ExponentialFogParameters2.w));
RayOriginTermsSecond = ExponentialFogParameters2.z * exp2(-
ExponentSecond);
}
// 雾厚度线积分
// Calculate the "shared" line integral (this term is also used for the
directional light inscattering) by adding the two line integrals
together (from two different height falloffs and densities)
// ExponentialFogParameters.y : fog height falloff
float ExponentialHeightLineIntegralShared =
CalculateLineIntegralShared(ExponentialFogParameters.y, RayDirectionY,
RayOriginTerms)
+ CalculateLineIntegralShared(ExponentialFogParameters2.y,
RayDirectionY, RayOriginTermsSecond);
// fog amount, 最终的积分值
float ExponentialHeightLineIntegral =
ExponentialHeightLineIntegralShared * RayLength;

// 雾色
half3 InscatteringColor = ExponentialFogColorParameter.xyz;
half3 DirectionalInscattering = 0;

// if InscatteringLightDirection.w is negative then it's disabled,
otherwise it holds directional inscattering start distance
if (InscatteringLightDirection.w >= 0)
{
    float DirectionalInscatteringStartDistance =
InscatteringLightDirection.w;
    // Setup a cosine lobe around the light direction to approximate
inscattering from the directional light off of the ambient haze;
    half3 DirectionalLightInscattering =
DirectionalInscatteringColor.xyz *
pow(saturate(dot(CameraToReceiverNormalized,
InscatteringLightDirection.xyz))), DirectionalInscatteringColor.w);

    // Calculate the line integral of the eye ray through the haze,
using a special starting distance to limit the inscattering to the
distance
    float DirExponentialHeightLineIntegral =
ExponentialHeightLineIntegralShared * max(RayLength -
DirectionalInscatteringStartDistance, 0.0f);
    // Calculate the amount of light that made it through the fog using
the transmission equation
    half DirectionalInscatteringFogFactor = saturate(exp2(-
DirExponentialHeightLineIntegral));
    // Final inscattering from the light

```

```

        DirectionalInscattering = DirectionalLightInscattering * (1 -
DirectionalInscatteringFogFactor);
    }

    // 比尔定律
    // Calculate the amount of light that made it through the fog using the
    transmission equation
    // 最终的系数
    half ExpFogFactor = max(saturate(exp2(-ExponentialHeightLineIntegral)),
MinFogOpacity);

    // ExponentialFogParameters3.w : FogCutoffDistance
    if (ExponentialFogParameters3.w > 0 && CameraToReceiverLength >
ExponentialFogParameters3.w)
    {
        ExpFogFactor = 1;
        DirectionalInscattering = 0;
    }

    half3 FogColor = (InscatteringColor) * (1 - ExpFogFactor) +
DirectionalInscattering;

    return half4(FogColor, ExpFogFactor);
}

```

3. 屏幕空间绘制，通过全屏绘制，避免水面和水平下的雾效重复绘制。使用drawmesh替换blit，避免切换RT带来的高额开销。