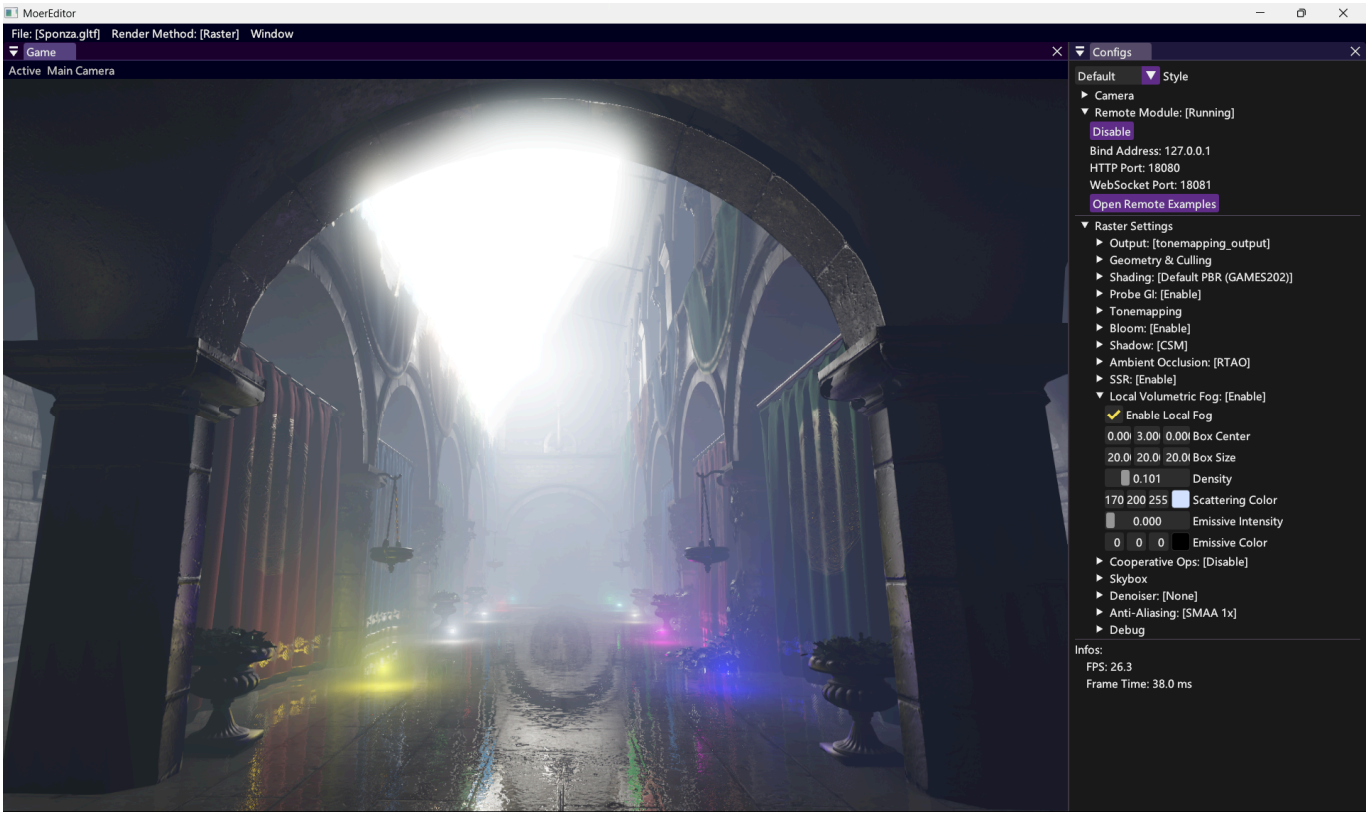
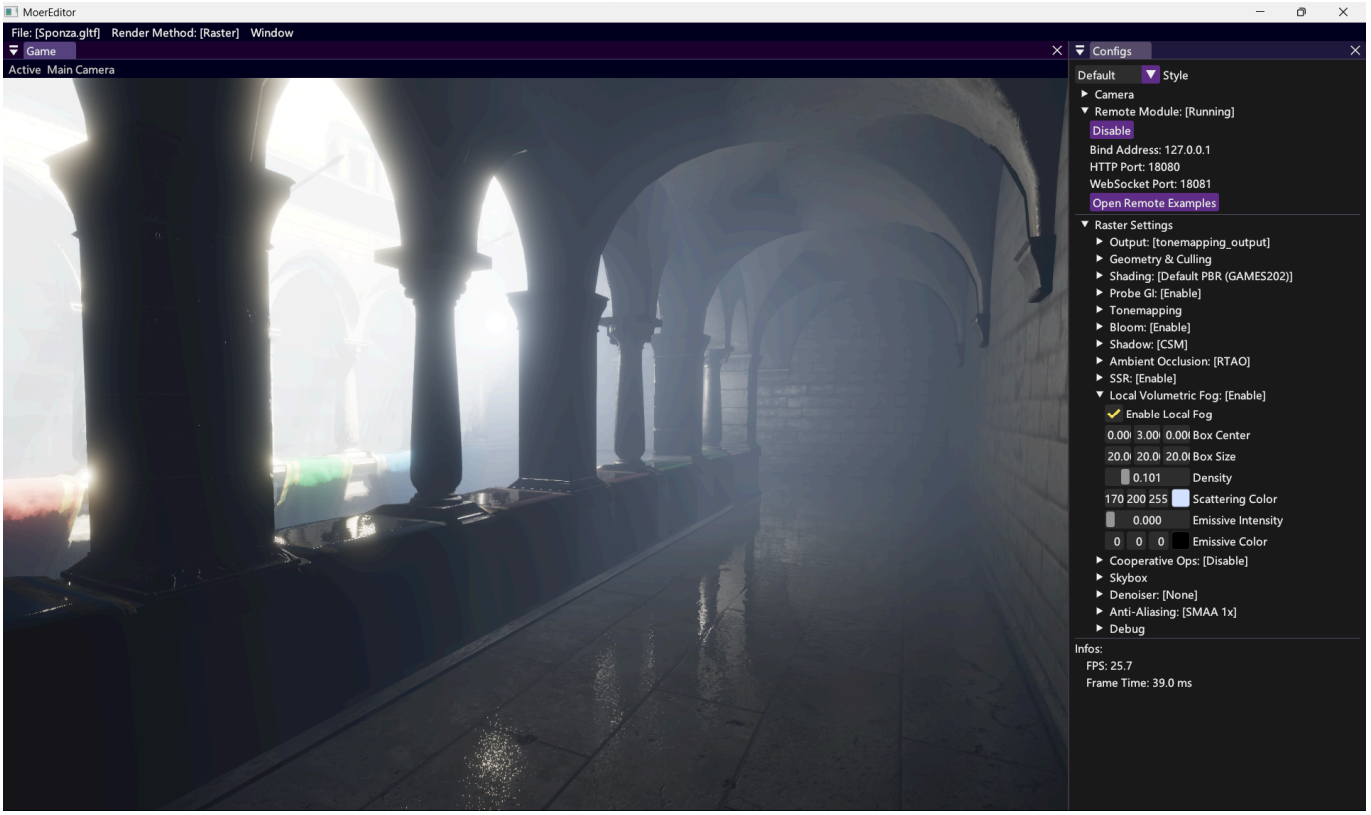
图形绘制技术

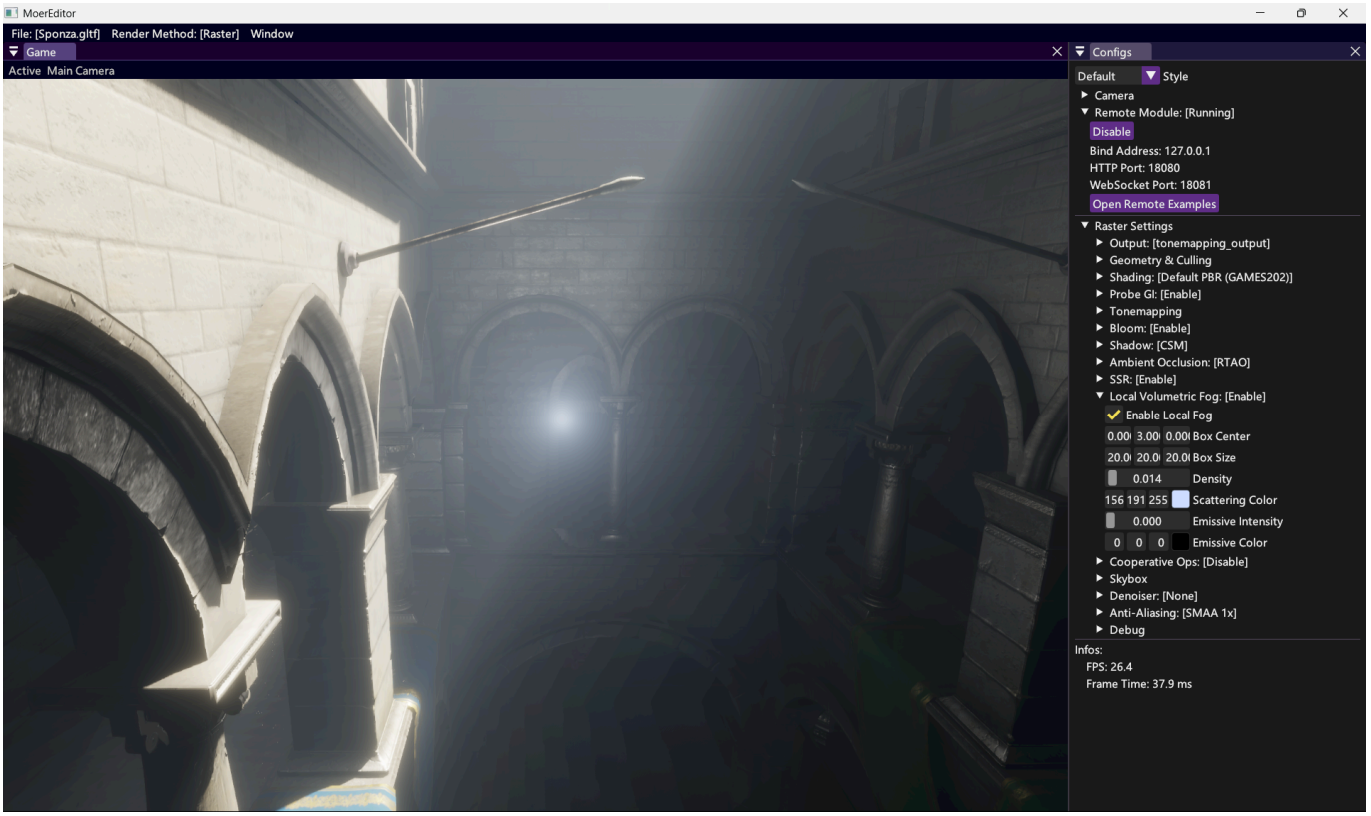
项目	学号	姓名
大作业-体积渲染	[redacted]	潘同学

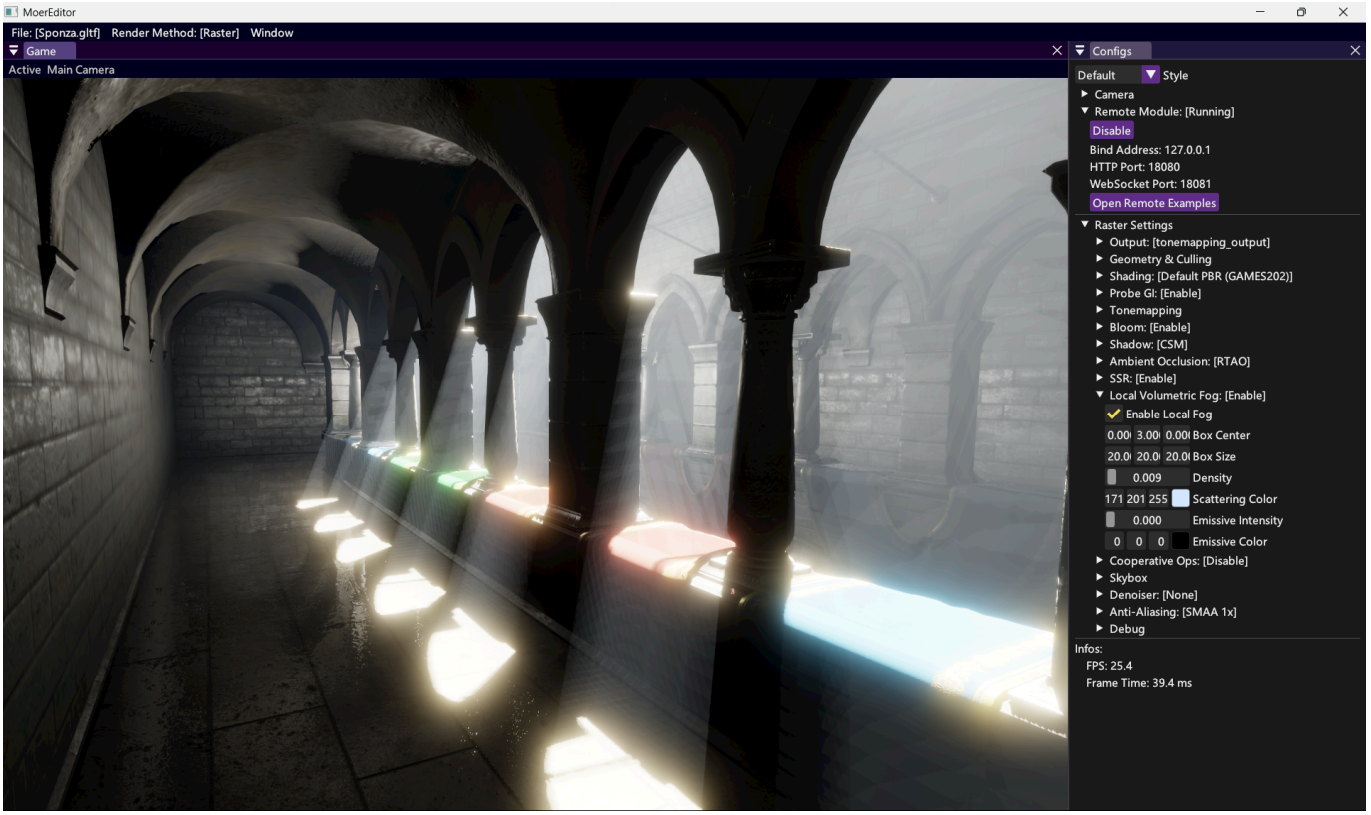
一、效果展示

包含点光源、平行光源的体积雾效果演示。




MoerEditor
File: [Sponza.gltf] Render Method: [Raster] Window
Game
Active Main Camera
Configs
Default Style
Camera
Remote Module: [Running]
Disable
Bind Address: 127.0.0.1
HTTP Port: 18080
WebSocket Port: 18081
Open Remote Examples
Raster Settings
Output: [tonemapping_output]
Geometry & Culling
Shading: [Default PBR (GAMES202)]
Probe GI: [Enable]
Tonemapping
Bloom: [Enable]
Shadow: [CSM]
Ambient Occlusion: [RTAO]
SSR: [Enable]
Local Volumetric Fog: [Enable]
Enable Local Fog
0.00 3.00 0.00 Box Center
20.0 20.0 20.0 Box Size
0.101 Density
170 200 255 Scattering Color
0.000 Emissive Intensity
0 0 0 Emissive Color
Cooperative Ops: [Disable]
Skybox
Denoiser: [None]
Anti-Aliasing: [SMAA 1x]
Debug
Infos:
FPS: 25.7
Frame Time: 39.0 ms


MoerEditor
File: [Sponza.gltf] Render Method: [Raster] Window
Game
Active Main Camera
Configs
Default Style
Camera
Remote Module: [Running]
Disable
Bind Address: 127.0.0.1
HTTP Port: 18080
WebSocket Port: 18081
Open Remote Examples
Raster Settings
Output: [tonemapping_output]
Geometry & Culling
Shading: [Default PBR (GAMES202)]
Probe GI: [Enable]
Tonemapping
Bloom: [Enable]
Shadow: [CSM]
Ambient Occlusion: [RTAO]
SSR: [Enable]
Local Volumetric Fog: [Enable]
Enable Local Fog
0.00 3.00 0.00 Box Center
20.0 20.0 20.0 Box Size
0.014 Density
156 191 255 Scattering Color
0.000 Emissive Intensity
0 0 0 Emissive Color
Cooperative Ops: [Disable]
Skybox
Denoiser: [None]
Anti-Aliasing: [SMAA 1x]
Debug
Infos:
FPS: 26.4
Frame Time: 37.9 ms



二、原理分析

1. 体积渲染方程

体积渲染需要求解参与介质中的辐射传输方程。只考虑单次散射时，视线方向的辐射亮度由衰减后的场景表面辐射与沿路径累积的介质散射两部分组成：

$$L(\mathbf{x}, \vec{\omega}) = T(s_{\text{surf}})\, L_{\text{surf}}(\mathbf{x}_{\text{surf}}, \vec{\omega}) + \int_0^{s_{\text{surf}}} T(s)\, J(\mathbf{x}(s), \vec{\omega})\, \mathrm{d}s$$

其中

- $\mathbf{x}(s) = \mathbf{c} + s \cdot \vec{\omega}$ 是沿视线方向的采样点，$\mathbf{c}$ 为相机位置，$\vec{\omega}$ 为视线方向
- $T(s) = \exp\left(-\int_0^s \sigma_e(\mathbf{x}(t))\, \mathrm{d}t\right)$ 为透射率，描述从距离 s 处到眼点的衰减
- σ_e 为消光系数，控制介质的不透明度
- $J(\mathbf{x}, \vec{\omega})$ 为源项，包含散射光和自发光
- s_{surf} 为场景表面到相机的距离

源项 J 的散射部分进一步展开为：

$$J(\mathbf{x}, \vec{\omega}) = \sigma_s(\mathbf{x}) \int_{S^2} p(\mathbf{x}, \vec{\omega}, \vec{\omega}')\, L_{\text{in}}(\mathbf{x}, \vec{\omega}')\, \mathrm{d}\vec{\omega}' + \sigma_a(\mathbf{x})\, L_e(\mathbf{x}, \vec{\omega})$$

其中 σ_s 为散射系数，p 为相位函数，L_{in} 为入射辐射，L_e 为自发光辐射。

2. Beer-Lambert 消光定律

介质衰减遵循 Beer-Lambert 定律，出射辐射与入射辐射呈指数关系：

$$L_{\text{out}} = L_{\text{in}} \cdot e^{-\sigma_e\, d}$$

3. 光线步进

上述积分采用光线步进算法近似求解。简易做法不考虑性能优化，故将路径等分为 N 步，步长 $\Delta t = (t_{\text{end}} - t_{\text{start}})/N$，并采用误差更小的中点采样：

$$t_i = t_{\text{start}} + \left(i + \frac{1}{2}\right)\Delta t, \quad i = 0, 1, \ldots, N-1$$

离散化后每一步的累积公式为：

$$C = T_N \cdot C_{\text{scene}} + \sum_{i=0}^{N-1} T_i \cdot \mathbf{S}_i \cdot \Delta t$$

$$T_{i+1} = T_i \cdot e^{-\sigma_e \, \Delta t}, \quad T_0 = 1$$

T_i 为第 i 步起点的累积透射率，$\mathbf{S}_i = \sigma_s \cdot L_{\text{scattered}}(\mathbf{x}_i) + \mathbf{E}$ 为该步中点的源项。其中 $L_{\text{scattered}}(\mathbf{x}_i)$ 是该采样点接收到的散射光总量——遍历所有光源求和，每个光源的贡献通过 Henyey-Greenstein 相位函数加权，详见下文。

σ_e 越大，雾越浓，透射率下降越快。

4. 散射光源计算与 Henyey-Greenstein 相位函数

散射光总量 $L_{\text{scattered}}(\mathbf{x}_i)$ 的计算流程为：遍历场景中所有光源，对每个光源取其贡献值乘以相位函数，加权求和。着色器支持方向光和点光源两种类型：

方向光：光源无穷远，所有采样点的光照方向 $\mathbf{L}$ 相同，无距离衰减。通过级联阴影贴图（CSM）确定阴影——CSM 将视锥体按深度划分为多个 Cascade，采样点根据视图深度选择对应级联查询阴影：

```
// 方向光
float3 L = normalize(-light.direction.xyz);
float shadow = SampleDirectionalShadow(sample_pos);
float phase = HenyeyGreensteinPhase(dot(ray_dir, L), param.anisotropy);
accumulated += light.color * light.intensity * shadow * phase;
```

点光源：光源具有空间位置，$\mathbf{L}$ 逐采样点计算。强度遵循平方反比定律 $I/(4\pi r^2)$，靠近光源的雾更亮。出于性能考虑未添加阴影：

```
// 点光源
float3 to_light = light.position.xyz - sample_pos;
float dist2 = max(dot(to_light, to_light), 1e-4);
float3 L = to_light * rsqrt(dist2);          // 归一化方向
float attenuation = 1.0 / (4.0 * PI * dist2); // 平方反比定律
float phase = HenyeyGreensteinPhase(dot(ray_dir, L), param.anisotropy);
accumulated += light.color * light.intensity * attenuation * phase;
```

两类光源均乘以 `HenyeyGreensteinPhase(dot(ray_dir, L), g)`，这即是上一节辐射传输方程中的相位函数 p。它描述了从光源方向 $\mathbf{L}$ 来的光被散射到视线方向 `ray_dir` 的概率分布：

$$p(\theta, g) = \frac{1}{4\pi} \cdot \frac{1 - g^2}{(1 + g^2 - 2g\cos\theta)^{3/2}}$$

其中 $\cos\theta = \vec{\omega}_{\text{view}} \cdot \vec{\omega}_{\text{light}} = \text{dot}(\text{ray\_dir}, \mathbf{L})$。$g \in [-1, 1]$ 为各向异性参数：$g > 0$ 为前向散射（水雾 $g \approx 0.7 \sim 0.9$），朝向光源看雾更亮；$g = 0$ 为各向同性散射，退化为常数；$g < 0$ 为后向散射。

σ_s 散射系数，从消光系数得到：$\sigma_s = \sigma_e \cdot \text{scattering\_color}$，决定散射 albedo。$\mathbf{E} = \text{emissive\_color} \cdot \text{emissive\_intensity}$ 为自发光项，普通雾设为零。

三、关键代码

主函数：体积雾积分循环

```
float4 main(float2 uv : TEXCOORD0) : SV_TARGET {
    // 1. 采样场景颜色与深度
    float3 scene_color = TextureHandle(param.input_image).Sample2D<float3>(uv);
    float depth = TextureHandle(param.depth_tex).Sample2D<float>(uv);

    // 2. 计算视线方向向量
    float3 camera_pos = lighting_data.camera_position;
    float3 far_world_pos = WorldPosFromDepth(0.0, uv, lighting_data.clip2world);
    float3 ray_dir = normalize(far_world_pos - camera_pos);
    // WorldPosFromDepth: 对深度值为0（远平面）的NDC坐标进行clip2world逆变换
    // far_world_pos - camera_pos 即为从相机穿过该像素射向场景的视线方向

    // 3. 构造体积雾包围盒，利用Slab Method求交点
    float3 half_size = max(param.volume_size * 0.5, float3(0.01, 0.01, 0.01));
    float3 box_min = param.volume_center - half_size;
    float3 box_max = param.volume_center + half_size;
    float t_enter, t_exit;
    if (!RayBoxIntersect(camera_pos, ray_dir, box_min, box_max, t_enter, t_exit))
        return float4(scene_color, 1.0);  // 视线与雾盒无交集，直接返回场景色

    // 4. 用场景深度约束积分终点（场景表面后的雾不可见）
    float surface_distance = 1e20;
    if (depth > 0.0) {
        float3 surface_world_pos = WorldPosFromDepth(depth, uv,
lighting_data.clip2world);
        surface_distance = distance(surface_world_pos, camera_pos);
    }
    float t_start = max(t_enter, 0.0);
    float t_end = min(t_exit, surface_distance);
    if (t_end <= t_start) return float4(scene_color, 1.0);

    // 5. 初始化步进参数
    uint step_count = max(param.step_count, 1u);
    float step_length = (t_end - t_start) / step_count;
    float extinction = max(param.density, 0.0);
    float3 sigma_s = extinction * max(param.scattering_color, float3(0.0, 0.0,
0.0));
    float3 emission = max(param.emissive_color, float3(0.0, 0.0, 0.0))
                    * max(param.emissive_intensity, 0.0);
```

```
    // 6. 沿射线步进，累积散射光
    float transmittance = 1.0;
    float3 accumulated = float3(0.0, 0.0, 0.0);
    [loop]
    for (uint step_index = 0; step_index < step_count; ++step_index) {
        float t = t_start + (step_index + 0.5) * step_length;  // 中点法则采样
        float3 sample_pos = camera_pos + ray_dir * t;

        float3 incoming = EvaluateScatteredLight(sample_pos, ray_dir);
        float3 source = sigma_s * incoming + emission;
        accumulated += transmittance * source * step_length;     // 当前步的贡献
        transmittance *= exp(-extinction * step_length);         // Beer-Lambert衰
减
    }

    // 7. 合成：衰减后的场景 + 累积的散射光
    float3 color = scene_color * transmittance + accumulated;
    return float4(color, 1.0);
}
```

Henyey-Greenstein 相位函数

```
float HenyeyGreensteinPhase(float cos_theta, float g) {
    float g2 = g * g;
    float denom = max(1.0 + g2 - 2.0 * g * cos_theta, 1e-4);
    return (1.0 - g2) / (4.0 * PI * denom * sqrt(denom));
}
```

EvaluateScatteredLight：散射光源累积

```
float3 EvaluateScatteredLight(float3 sample_pos, float3 ray_dir) {
    ArrayBuffer light_buf = ArrayBuffer(param.light_buf_hdl);
    float3 accumulated = float3(0.0, 0.0, 0.0);

    [loop]
    for (uint light_index = 0; light_index < lighting_data.light_count;
++light_index) {
        Moer::GLight light = light_buf.Load<Moer::GLight>(light_index);

        if (light.type == Moer::ELightType::Directional) {
            // 平行光：L 为常量方向，无距离衰减，有CSM阴影
            float3 L = normalize(-light.direction.xyz);
            float shadow = SampleDirectionalShadow(sample_pos);
            float phase = HenyeyGreensteinPhase(dot(ray_dir, L),
param.anisotropy);
            accumulated += light.color * light.intensity * shadow * phase;
        }
        else if (light.type == Moer::ELightType::Point) {
```

```
            // 点光源：L 随采样点位置变化，平方反比衰减，无阴影
            float3 to_light = light.position.xyz - sample_pos;
            float dist2 = max(dot(to_light, to_light), 1e-4);
            float3 L = to_light * rsqrt(dist2);          // 归一化方向
            float attenuation = 1.0 / (4.0 * PI * dist2); // 平方反比定律
            float phase = HenyeyGreensteinPhase(dot(ray_dir, L),
param.anisotropy);
            accumulated += light.color * light.intensity * attenuation * phase;
        }
    }
    return accumulated;
}
```

CSM 方向光阴影查询

```
float SampleDirectionalShadow(float3 world_pos) {
    // 非CSM模式则不计算阴影
    if (lighting_data.shadow_map_mode != Moer::EShadowMapMode::CSM
        && lighting_data.shadow_map_mode != Moer::EShadowMapMode::CSM_AUTO)
        return 1.0;

    // 根据采样点的视图深度选择对应的Cascade级联
    float pixel_view_z = 0.0;
    int cascade_index = get_cascade_index(lighting_data, world_pos, pixel_view_z);
    if (cascade_index < 0) return 1.0;  // 超出所有级联范围

    // 将采样点变换到该级联的阴影贴图空间
    float4 shadow_clip_pos = mul(
        lighting_data.world2shadow_clip[cascade_index], float4(world_pos, 1.0));
    float3 shadow_ndc_pos = shadow_clip_pos.xyz / shadow_clip_pos.w;
    float2 shadow_uv = float2(
        shadow_ndc_pos.x * 0.5 + 0.5,
        1.0 - (shadow_ndc_pos.y * 0.5 + 0.5));

    // 范围检查：超出阴影贴图范围的区域视为无阴影
    bool in_bounds = shadow_uv.x >= 0.0 && shadow_uv.x <= 1.0
                  && shadow_uv.y >= 0.0 && shadow_uv.y <= 1.0
                  && shadow_ndc_pos.z >= 0.0 && shadow_ndc_pos.z <= 1.0;
    if (!in_bounds) return 1.0;

    // Reverse-Z 深度比较：存储深度更大表示采样点处于遮挡区域
    float stored_depth = TextureHandle(
        lighting_data.cascade_shadow_map[cascade_index]).Sample2D<float>
(shadow_uv);
    const float shadow_bias = 3e-4;
    return stored_depth > shadow_ndc_pos.z + shadow_bias ? 0.0 : 1.0;
}
```

四、工程框架

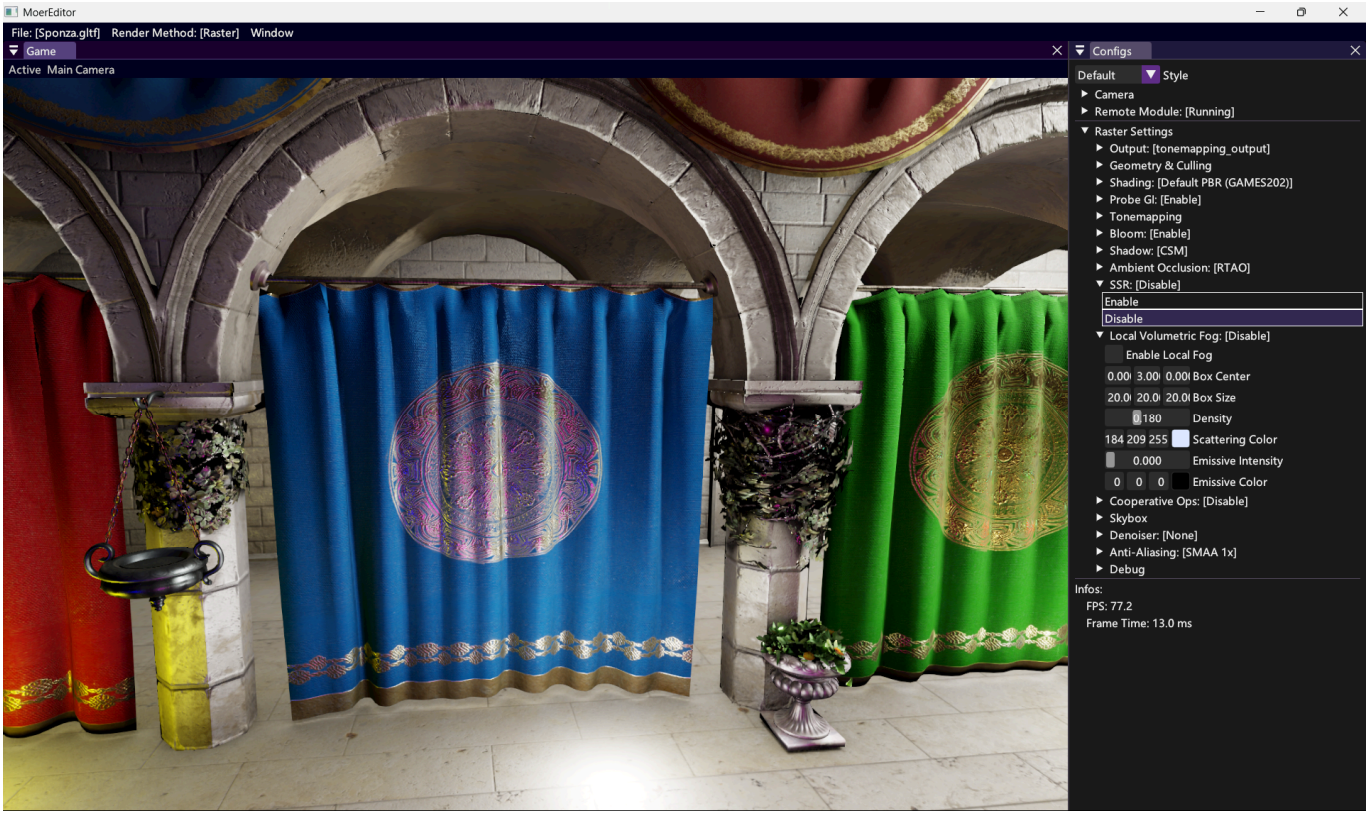
本项目在 MoerEngine 现有光栅化渲染框架中增加了一个局部体积雾后处理阶段（插入到光照、屏幕空间效果和抗锯齿之间）。仿照其他pass，除了着色器 `VolumetricFog.hlsl` 外，主要完成了以下引擎框架的改动：

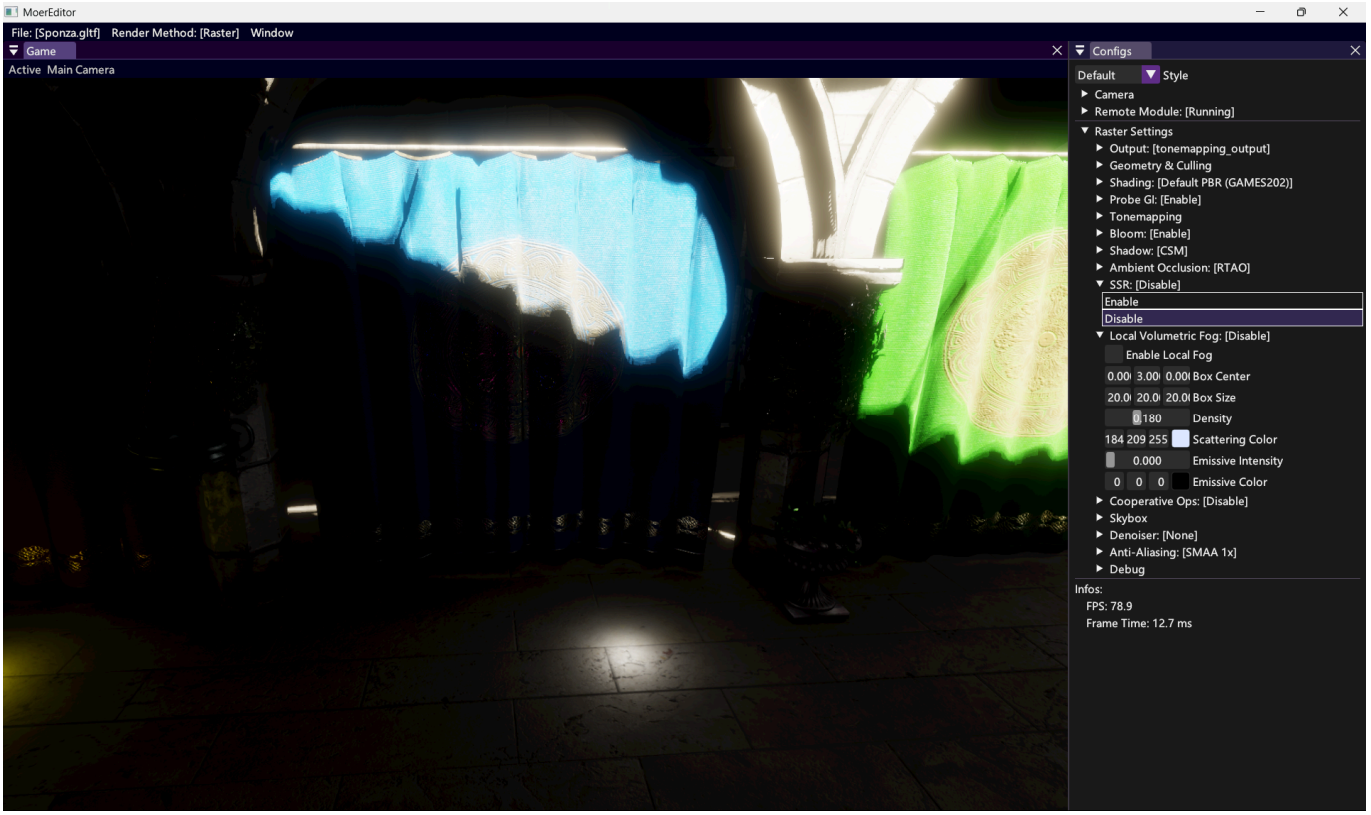
- 在 `RasterConfig` 中增加体积雾配置，包括启用开关、雾密度、散射颜色、自发光颜色和强度、各向异性参数、光线步进次数，以及局部雾盒的中心位置和长宽高（仅实现了单一雾盒）。
- 增加参数结构 `VolumetricFogPipelineBindlessParam`，向Shader传递雾盒参数、光照缓冲区、场景颜色纹理和深度纹理。
- 新增 `VolumetricFogPass`，负责创建全屏光栅化 Pipeline、绑定光照数据和 Bindless 资源，并将体积雾计算结果写入新的 `volumetric_output` HDR 纹理。
- 在UI界面中增加 `Local Volumetric Fog` 选项设置，支持启用或禁用体积雾、输入雾盒中心和尺寸、调整密度、散射颜色、自发光颜色及自发光强度。

五、课程总结

课程的实验非常有趣，能够带领学生探索MoerEngine等贴近工业实际的渲染引擎的源码，清晰地看到之前仅有基本概念的图形算法是怎么实现的。环境配置引导也做得比较好，跟着操作做基本不会遇到编译问题。没有什么别的建议。

对MoerEngine光栅化渲染器的使用反馈：Bloom效果现在只有开启和关闭的选项，可以增加强度控制。在sponza场景中发现有随视角转动变化的漏光问题，原因未知。





上面这里本来是处在阴影当中，不应该有平行光打过来（如第一张图），但是视角往左边转一下，右上角区域就出现漏光了。