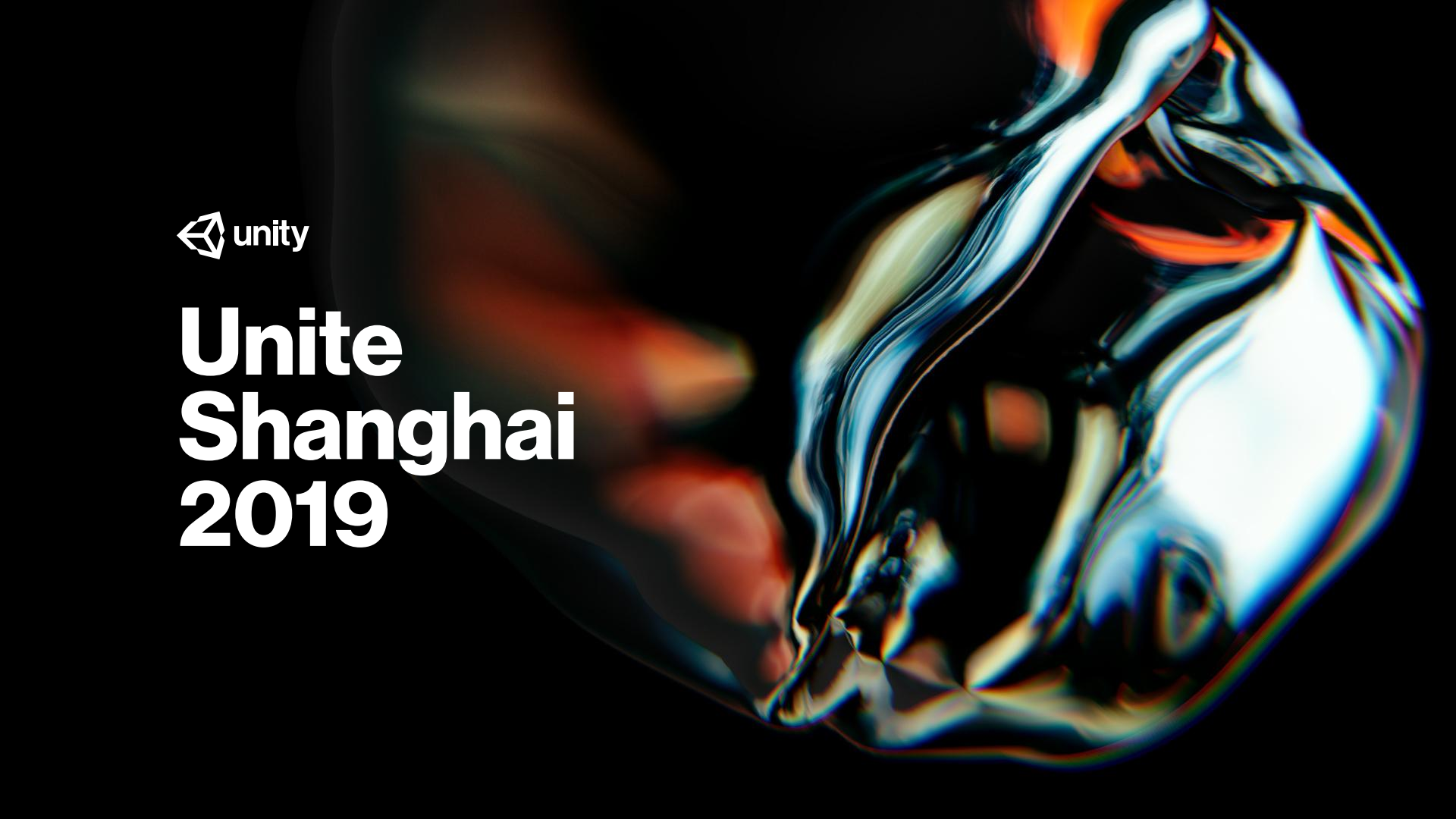
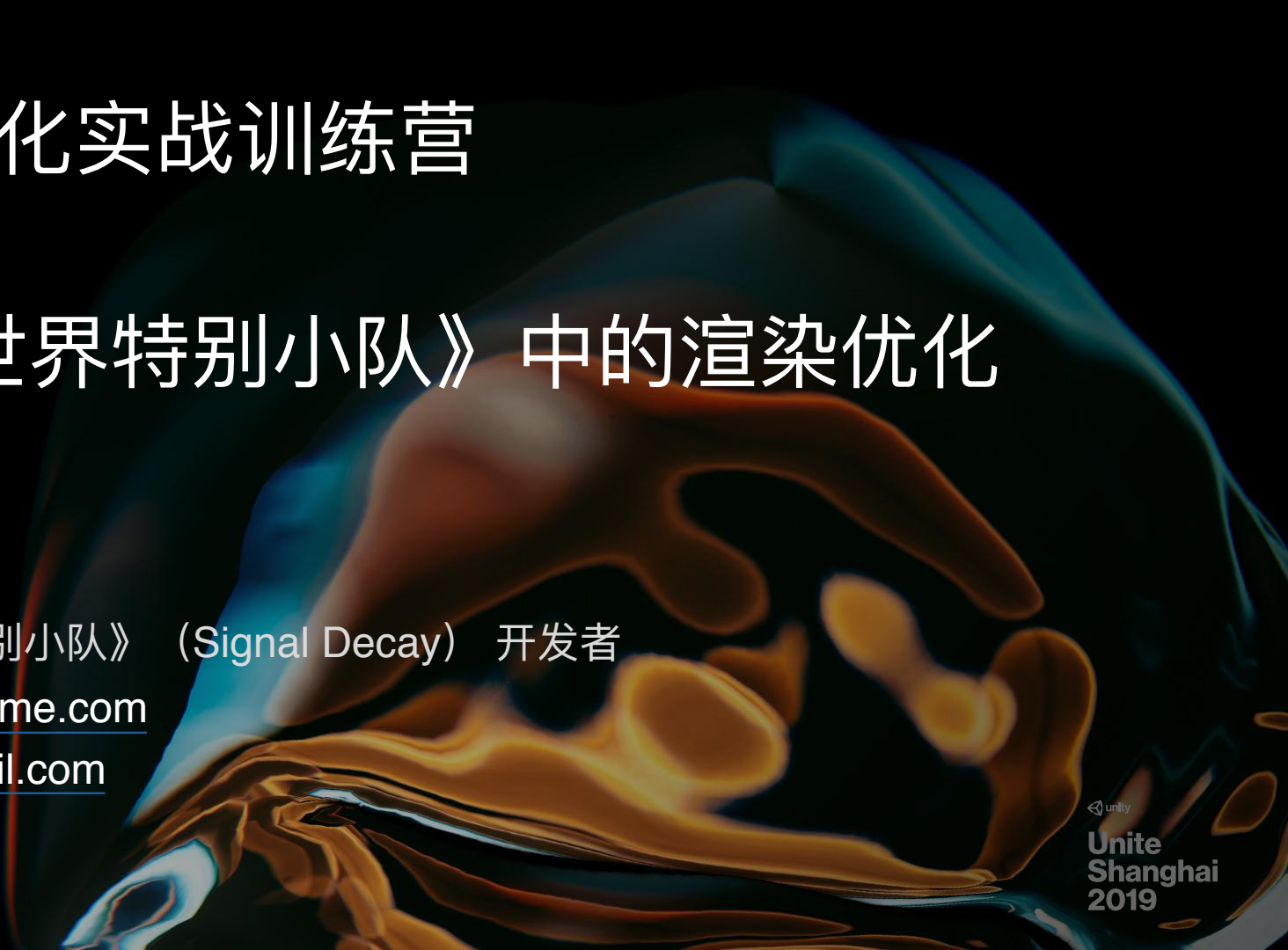

unity
Unite
Shanghai
2019

Unity 优化实战训练营

《拯救世界特别小队》中的渲染优化

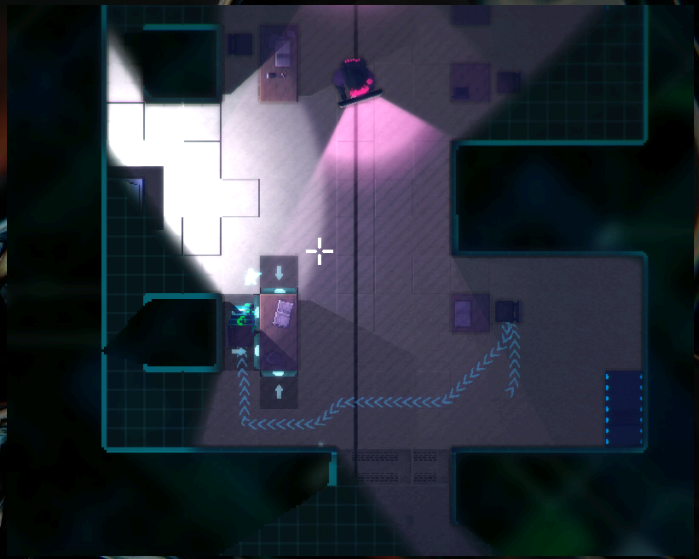
Zack
《拯救世界特别小队》（Signal Decay） 开发者
signaldecaygame.com
zacx.zh@gmail.com



unity
Unite Shanghai 2019

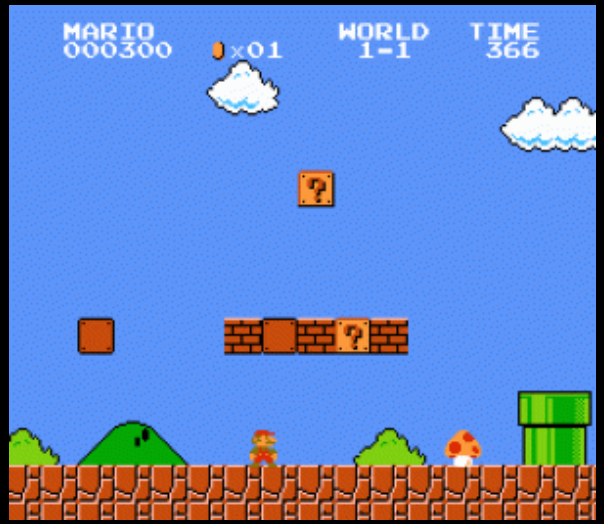
实践内容

- 使用 Tilemap 优化工作流和提升效率
- 定制 2D 游戏的延迟光照优化光影渲染
- 运用 GPU 优化游戏的动态视野效果渲染

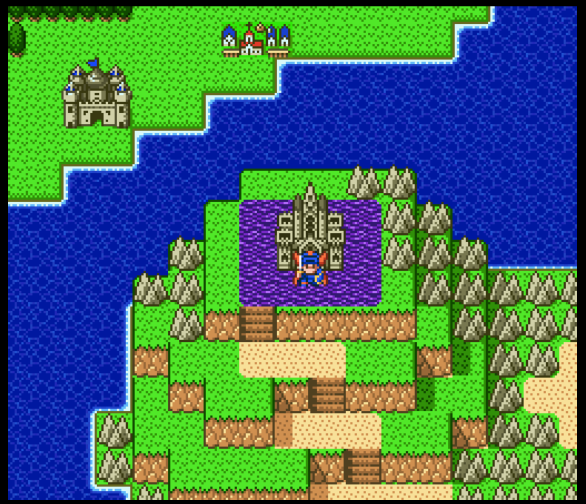


Tilemap 在项目中的实践运用

2D 游戏 Tilemap



超级马里奥



勇者斗恶龙

在《拯救世界特别小队》中使用 Tilemap

- 使用 Tilemap 的优势
    - 简化工作流
    - 减少渲染批次
    - 容易进行后续优化
- 其他需求
    - Tile 表示的物件可以产生复杂的交互
    - 让 tilemap 支持光照

批量创建 tiles

- 找到 *Assets/Entities/Stage/Tilesets/floor01.png*
- 右键 选择 Create/Create Tiles From Sprite
- 选择 *Assets/Entities/Stage/Tiles* 目录 工具会自动从 sprites 创建 tiles。

- 菜单选中 Window > 2D > Tile Palette
- 在 Tile Palatte 的下拉菜单中选中 Create New Palette。在新弹出的菜单中命名为 Default Palette，可保存在 Assets 文件夹下。
- 在 Project 窗口中找到选中所有刚刚创建好的 tiles，拖入 Tile Palatte 窗口

绘制 tilemap

- 打开 *Assets/Scenes/Mission.unity*
- 在 Hierarchy 窗口中选择 Level 物体
- 在 Hierarchy 窗口右键 Level 物体，选择 Create Empty。将新物体命名为 Floor Tilemap。将其 z 坐标设为 2。
- 为 Floor Tilemap 物体添加 Tilemap 和 Tilemap Renderer 组件。
- 打开 Tile Palette 窗口，选择笔刷工具。目标选择 Floor Tilemap。可以选中想要的 tile 在 Scene 窗口进行绘制。

自定义 Tile 类

- 配置 Instanced GameObjects 以实现复杂的交互
- 为 Tile 增加自定义属性

自定义 Tile 类

- 打开 *Assets/Scripts/Tiles/TileWithObject.cs* 文件
- 添加如下代码：

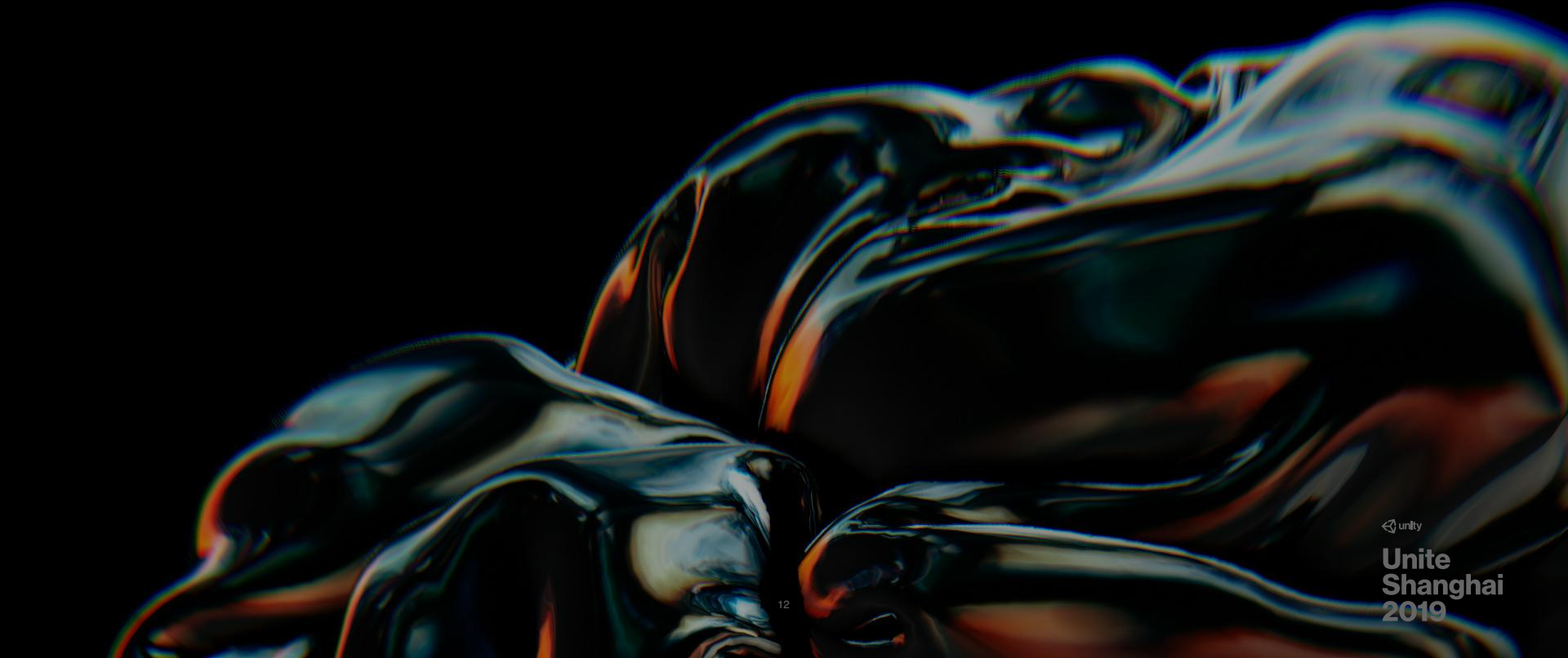
```
7       // EXERCISE support lit2D
8       public Texture2D normalMap;
9       public Texture2D emissionMap;
10      // end EXERCISE
11      public override bool StartUp (Vector3Int position, ITilemap tilemap, GameObject go) {
```

- 在 Project 窗口中找到 *Assets/Entities/Stage/Tilesets/tileset.png*
- 右键 从下拉菜单中选择 Create/Create Object Tiles From Sprites。用上述方法创建 tiles。
- 找到生成的 tiles，全部选中。在 Project 窗口中找到 *Assets/Entities/Stage/Tilesets/Tileset_n.png*，拖动到 Inspector 窗口中的 Normal Map 属性；将 *Assets/Entities/Stage/Tilesets/Tileset_e.png* 设置到 emission map 属性

绘制基于物件的 Tilemap

- 将新 tile 加入到 tile palette 中
- 在 Hierarchy 窗口中 Level 物件右键 Create Empty，取名 Wall Tilemap。
- 在 Wall Tilemap 上加入 Tilemap 组件（不加 Tilemap Renderer）
- 将 Tile Palette 目标设置为 Wall Tilemap。选择刚刚创建的 tile。尝试绘制，会发现显示不出来
- 保存场景，在 Project 创建选中刚刚创建的多个 tile，将 *Assets/Entities/Stage/Prefabs/Wall.prefab* 拖入 Instanced Game Object 属性。
- 可能需要重新载入场景

定制 2D 延迟光照渲染



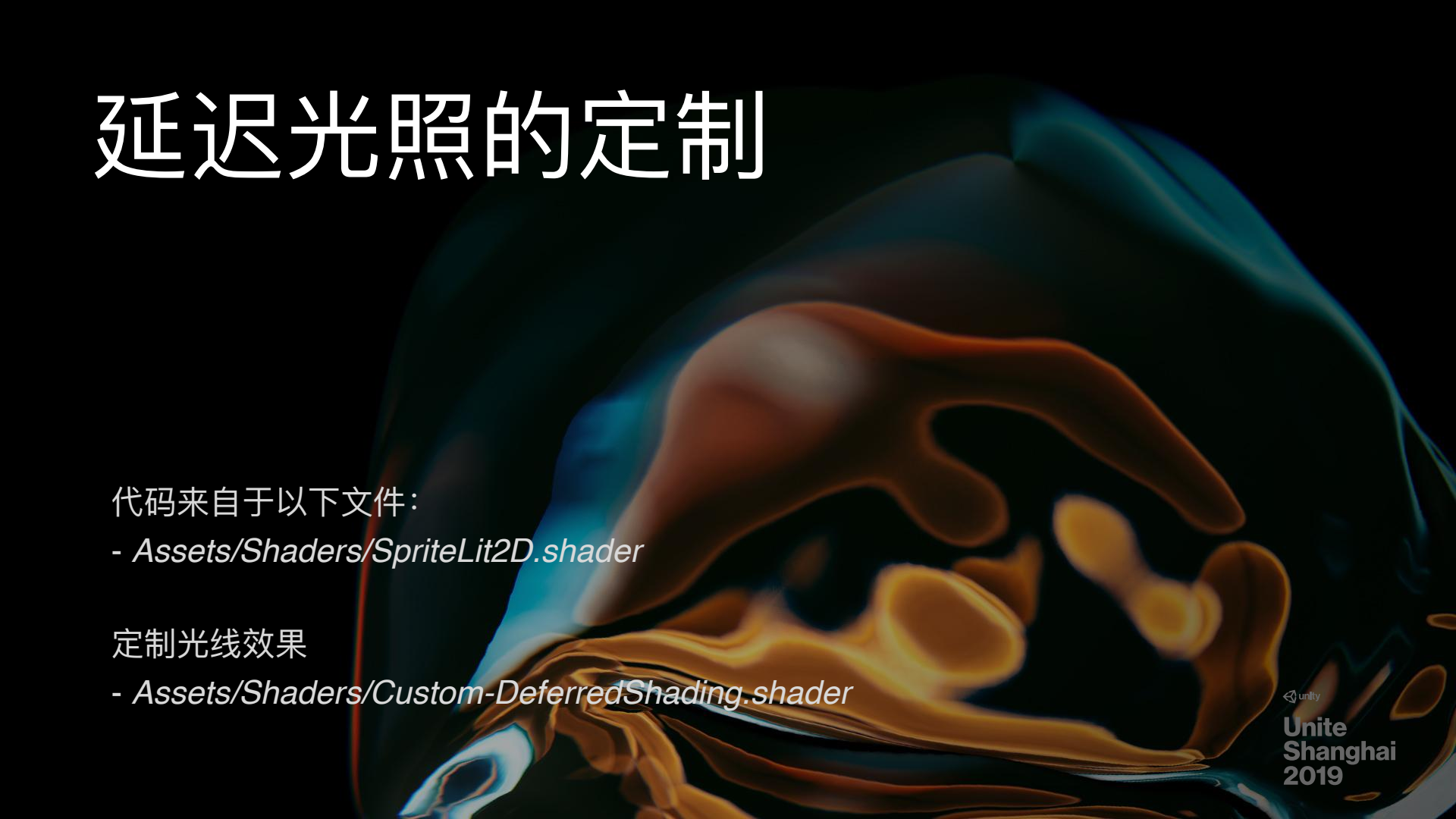
延迟光照管线与 Forward 管线的比较

- 优势
  - 多光源的场景中性能更高
  - 能正确渲染被多个光线照到的大物体的光影
- 劣势
  - 一些设备不支持
  - 一些功能受限（如占用 layer，stencil 在 base pass 期间不能用等）

为什么要自己定制

- 默认的光影材质不支持 Alpha Cutout
- 默认的光影材质不支持 PerRendererData。场景中来自不同纹理的 sprites 会需要/产生很多材质拷贝。使用 PerRendererData 可以合理优化内存
- 通过 hack 的方式，使 Unity 默认延迟光照管线支持正交投影摄像机。

延迟光照的定制



代码来自于以下文件：

- *Assets/Shaders/SpriteLit2D.shader*

定制光线效果

- *Assets/Shaders/Custom-DeferredShading.shader*

切换到延迟光照

- 打开 *Assets/Scenes/Example Mission.unity* 此时为 Forward 的 2D 画面
- 进入菜单 Edit/Project Setting。左边列表选择 Graphics 分页。
- 去掉 Tier Settings 所有的 Use Defaults 选项，将每个 Tier 的 Rendering Path 改为 Deferred
- 在 Scene 窗口上，取消 2D开关来预览延迟光照效果。可以看到画面里的精灵是黑色的。

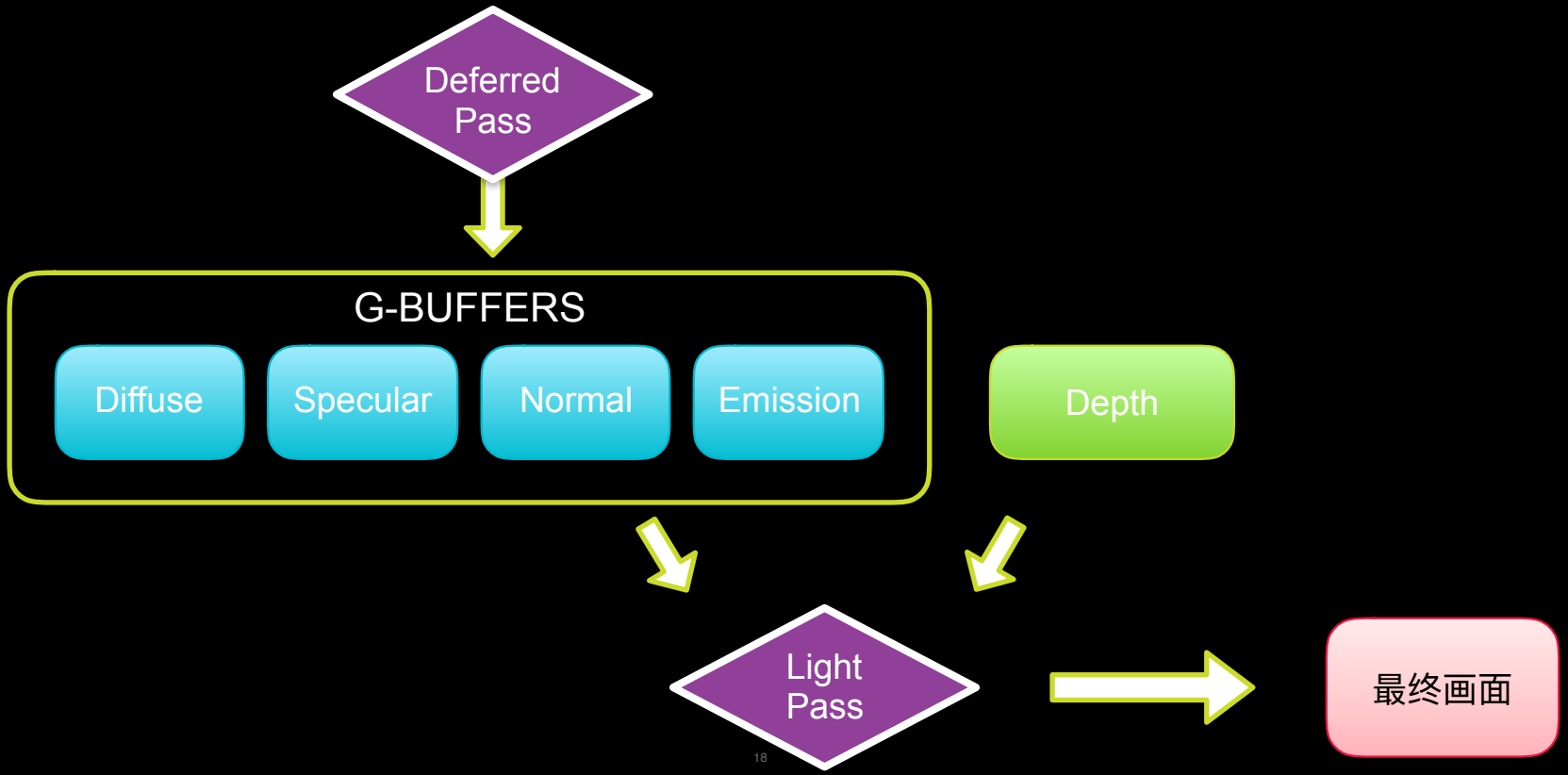
修改着色器代码

- 所有的精灵都使用 SpriteLit2D 的材质，其着色器代码在 *Assets/Shaders/SpriteLit2D.shader* 下
- 打开着色器文件，修改代码如右：

```
float3 normalWorld = GET_NORMAL_WORLD(input);
fixed4 emissionColor = tex2D(_EmissionMap, input.uv);
emissionColor.rgb *= emissionColor.a;
fixed3 finalEmission = emissionColor.rgb + unity_AmbientSky.rgb * unity_AmbientSky.a;

// EXERCISE
UnityStandardData data;
data.diffuseColor   = c.rgb;
data.occlusion      = 1;
data.specularColor  = 0;
data.smoothness     = _Shininess;
data.normalWorld    = normalWorld;

UnityStandardDataToGbuffer(data, output.albedo, output.specular, output.normal);

output.emission = fixed4(finalEmission, 1);
// end EXERCISE

return output;
```

- 进入 Unity 编辑器，会看到精灵图元具备基本的光照效果。

延迟光照原理


Deferred Pass
G-BUFFERS
Diffuse
Specular
Normal
Emission
Depth
Light Pass
最终画面

增加法线贴图和自发光贴图

- 注意到着色器文件的 NormalMap 和 EmissionMap 属性 被标注了 PerRendererData。我们可以通过 Renderer 的 Material Property Block 功能设置它们。
- 打开 *Assets/Scripts/Rendering/Lighting2D/SpriteLit2D.cs* 文件，添加右侧代码：

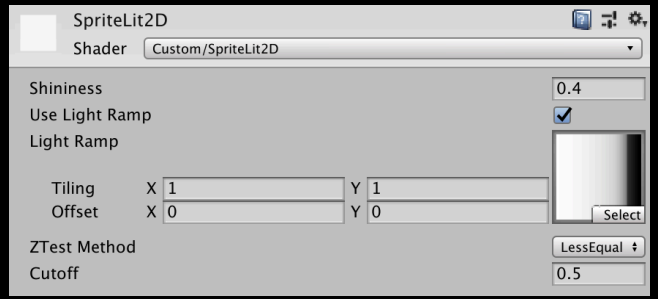
- 进入 Unity，从 Hierarchy 窗口中选择 Guard 打开 prefab，选择 Sprite 物件。在 Inspector 窗口中找到 SpriteLit2D 组件。
- 将 *Assets/Sprites/Guard_n.png* 拖入 Normal Map 属性中。
- 将 *Assets/Sprites/Guard_e.png* 拖入 Emission Map 属性中。
- 同理可将 Player 的 prefab 的 Sprite 物件加入贴图
  - 法线贴图 *Assets/Entities/Player/Sprites/MainChar01_n.png*
  - 自发光贴图 *Assets/Entities/Player/Sprites/MainChar01_e.png*

```
        return propertyBlock;
    }

    // EXERCISE sprite lit normal map
    public Texture2D normalMap;
    public Texture2D emissionMap;
    // end EXERCISE
    public Color emissionTint = Color.white;

    private MaterialPropertyBlock propertyBlock;
    private Renderer m_Renderer;



    public void ResetMaterialProperties () {
        if (m_Renderer == null) return;
        propertyBlock = GetDefaultPropertyBlock(m_Renderer);

        // EXERCISE sprite lit normal map
        if (normalMap != null) propertyBlock.SetTexture(NormalMapID, normalMap);
        if (emissionMap != null) propertyBlock.SetTexture(EmissionMapID,
emissionMap);
        // end EXERCISE

        m_Renderer.SetPropertyBlock(propertyBlock);
    }
```

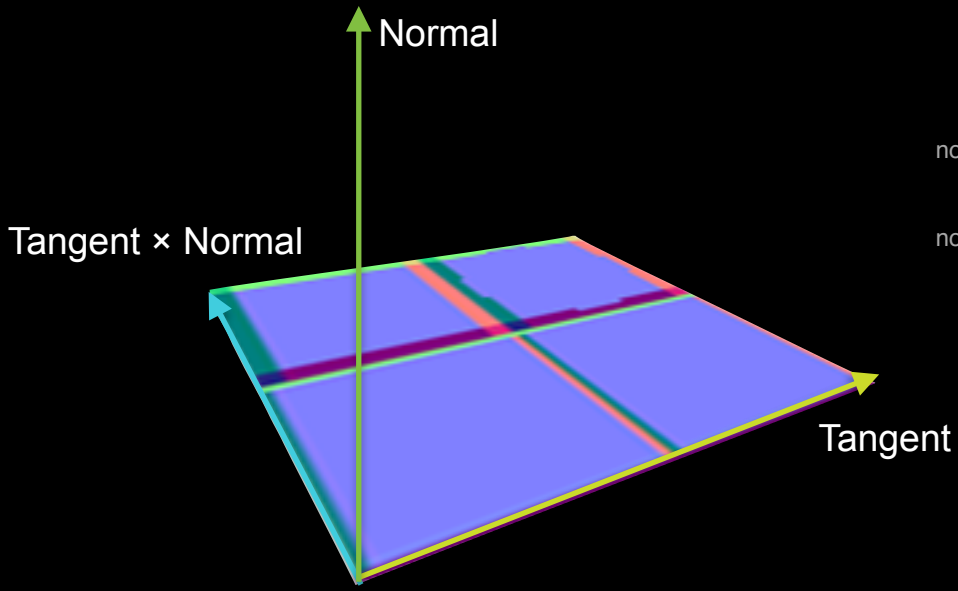
PerRendererData 和 Render Property Block

- PerRendererData 属性不会在材质界面中显示出来，无法设置在材质上
- 通过 Renderer.SetPropertyBlock/GetPropertyBlock 来设置
- 设置不同的属性值不会内部生成新的材质拷贝

```
[PerRendererData] _MainTex ("Texture", 2D) = "white" {}
[PerRendererData] _EmissionMap ("Texture", 2D) = "black" {}
[PerRendererData][Normal] _NormalMap ("Normal Map", 2D) = "bump" {}
[PerRendererData] _EmissionTint ("Emission Tint", Color) = (1, 1, 1, 1)
_Shininess ("Shininess", Float) = 1
[Toggle(LIGHTRAMP_ON)] _UseLightRamp ("Use Light Ramp", Float) = 0
_LightRamp ("Light Ramp", 2D) = "white" {}
[Enum(UnityEngine.Rendering.CompareFunction)] _ZTestMethod ("ZTest Method",
Float) = 4
_Cutoff ("Cutoff", Float) = 0.5
```



使用 Tangent 和 Normal 还原局部坐标




```
float3 ToObjectNormal (float3 tangentBase, float3 normalBase, float3
normal) {
    float3 otherBase = cross(tangentBase, normalBase);
    return normal.x * tangentBase + normal.y * otherBase + normal.z *
normalBase;
}
```

自定义光线效果

- 自定义 Lighting Pass：
  - 打开 Edit/Project Settings 在打开的窗口中选择 Graphics
  - 在下方 Built-in Shader Settings 中找到 Deferred 一栏，将下拉菜单选项从 Built-in shader 改成 Custom shader
  - 将 *Assets/Shaders/Custom-DeferredShading.shader* 拖入出现的 Shader 属性上
  - 可以看到游戏里的光效产生了变化
- 在 Project 窗口中找到 *Assets/Resources/DeferredRenderingSettings.asset*。可以看到 Lighting Ramp 属性是一张渐变纹理。
  - 将 *Assets/Misc/Lighting/* 下的不同的渐变纹理拖入到 Lighting Ramp 可以看到不同的光效。如果使用 Photoshop 的话可以自己做渐变纹理，实现不同的艺术效果。

为 Tilemap 增加光影效果

- 打开之前的 Mission.unity 文件
- 在 Hierarchy 窗口中选择 Floor Tilemap 物件。在 Inspector 窗口中找到 Tilemap Renderer 组件，将 *Assets/Materials/SpriteLit2D.mat* 材质拖入 Material 属性。
- 点击 Add Component。添加 SpriteLit2D 组件。将 floor02_n.png 拖入 Normal Map 属性。将 floor02_e 拖入 Emission Map 属性。

因为 API 暂时不支持（将于2019.3支持），带光影的 Tilemap 的所有 Tiles 必须来自同一纹理。

为 TileWithObject 支持光影效果

- 打开 *Assets/Scripts/Tiles/TileWithObject.cs* 文件
- 添加如下代码：

```
7       public override bool StartUp (Vector3Int position, ITilemap tilemap, GameObject go) {
8           if (go != null) {
9               Utils.InitializeTileObject(go, tilemap.GetComponent<Tilemap>(), tilemap.GetTransformMatrix(position), sprite);
10              // EXERCISE support lit2D
11              var ls = go.GetComponent<SpriteLit2D>();
12              ls.normalMap = normalMap;
13              ls.emissionMap = emissionMap;
14              // end EXERCISE
15          }
16          return base.StartUp(position, tilemap, go);;
17      }
```

- 重新载入 Mission.unity

支持正交投影 第一步

- 选择 Main Camera 物件，添加 Force Ortho Camera 组件
- 打开 *Assets/Scripts/Rendering/ForceOrthoCamera.cs*
- 找到如下的位置，添加右侧代码：

```
    private void UpdateCameraMatrix () {
        // EXERCISE set ortho matrix
        var cam = GetComponent<Camera>();
        cam.projectionMatrix = Matrix4x4.Ortho(-orthoSize * cam.aspect, orthoSize *
cam.aspect, -orthoSize, orthoSize, cam.nearClipPlane, cam.farClipPlane);
        // end EXERCISE
    }
```

- 这可以避免 Unity 自动切换到 Forward 管线实现正交投影的效果

支持正交投影 第二步

- 注意到此时的光影效果并不正确，这是因为标准延迟光照管线没考虑正交摄像机的结果
- 打开 *Assets/Shaders/Custom-DeferredShading.shader*
- 加入右侧代码：

```
half4 CalculateLight (unity_v2f_deferred i)
{
    float3 wpos;
    float2 uv;
    float atten, fadeDist;
    UnityLight light;
    UNITY_INITIALIZE_OUTPUT(UnityLight, light);
    // EXERCISE add ortho support
    if (unity_OrthoParams.w > 0) {
        OrthoDeferredCalculateLightParams (i, wpos, uv, light.dir, atten, fadeDist);
    } else {
        CustomDeferredCalculateLightParams (i, wpos, uv, light.dir, atten, fadeDist);
    }
    // end EXERCISE
```

- 进入编辑器，可以看到光影已修正

实现 GPU 加速的视野/视锥效果

常见的视野渲染算法 Raycast + 网格生成

来源：ncase.me

扇形视锥效果

- 打开 *Assets/Scenes/Mission.unity* 文件
- 在 Project 窗口中找到 *Assets/Prefabs/VisionCone.prefab* 文件，将其拖入到 Hierarchy 窗口中。调整到可以被镜头看见的位置，不要改变 z 坐标。
- 在 Hierarchy 窗口中选中 Main Camera 物件。在 Inspector 窗口中添加组件，选择 Render Sight 组件。
- 开始游戏，可以看到 Game 窗口下出现一个紫色的扇形。

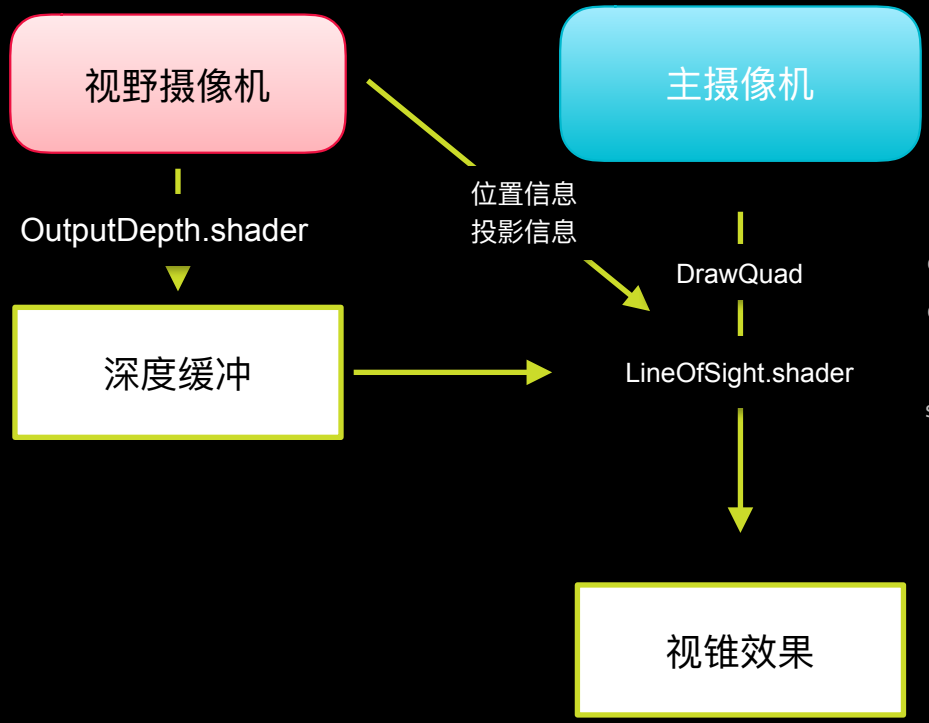
实现视野遮挡

- 选择 Vision Cone 物件，可以看到 Scene 窗口的 Camera Preview 窗口中是一块白色的画布。
- 打开 *Assets/Shaders/OutputDepth.shader* 文件，修改代码如右：

```
v2f vert( appdata_base v ) {
    v2f o;
    UNITY_SETUP_INSTANCE_ID(v);
    UNITY_INITIALIZE_VERTEX_OUTPUT_STEREO(o);
    o.pos = UnityObjectToClipPos(v.vertex);
    o.nz.xyz = COMPUTE_VIEW_NORMAL;
    // EXERCISE output depth
    o.nz.w = COMPUTE_DEPTH_01;
    // end EXERCISE
    return o;
}
```

- 回到 Unity，可以看到摄像机预览是一张深度图。同时 Game 窗口下的视野有了被遮挡的效果。

原理（阴影贴图原理）




```
o.worldPos = TransformClipToWorldPos(_InverseViewProjTransform,
o.vertex);

sightClipPos = mul(_SourceViewProjTransform, i.worldPos);
```

减少不必要的渲染

- 打开 *Assets/Prefabs/Wall.prefab*
- 在 Hierarchy 窗口下选中 Shadow Caster。在 Inspector 窗口中找到 Mesh Renderer 组件。
- 将 *Assets/Materials/Shadow Caster.mat* 拖入 Material 属性。
- 重新载入 *Assets/Scenes/Mission.unity* 可以看到墙的模型不会被渲染出来，但是可以投射阴影。

玩家视野的渲染

- 在 Project 窗口找到 *Assets/Prefabs/Player Sight Source.prefab* 将其拖入 Hierarchy 窗口中。在 Scene 窗口中将其调整到合适的位置。不要改变 z 坐标。
- 在 Hierarchy 窗口中选中 Main Camera，添加 Render Sight Panorama 组件。
- 可以看到 Game 窗口中显示了玩家的视野形状。
- 勾选 Render Sight Panorama 组件中的 Mask 选项，可以让视野以遮罩显现。

玩家视野渲染原理

- 可以采用四个方向的摄像机实现。
  - 这里只使用了一个摄像机，因为对场景进行了极坐标渲染。
- 极坐标不符合 GPU 插值的工作方式，因此这是近似渲染。
  - 运用了几何着色器校正插值
  - 只保证这种特殊场合的结果是正确的。