



中国移动开发者大会
Mobile Developer Conference China 2016



虚幻4渲染系统结构解析

房燕良 2016年9月

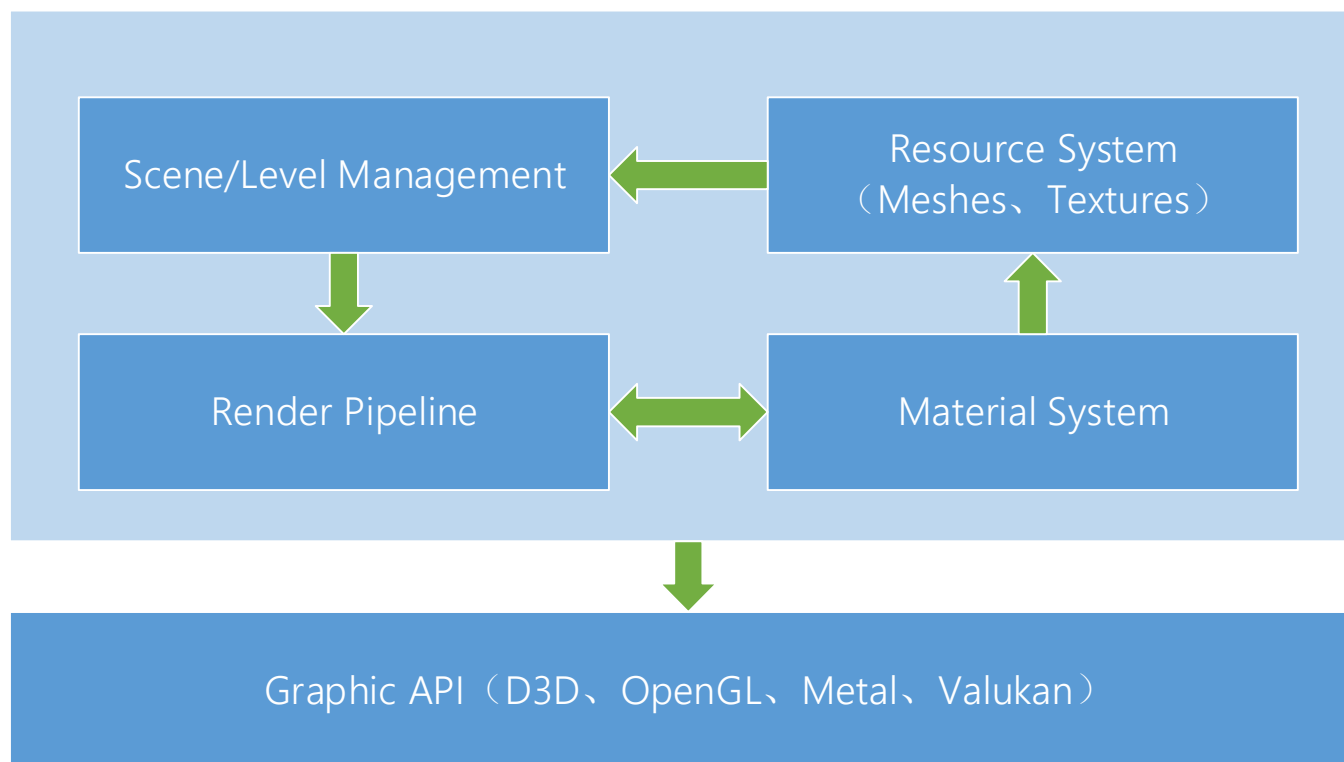
mdcc.csdn.net

- 自我介绍
- 3D引擎渲染系统概述
- 虚幻4渲染系统架构
 - 游戏线程&渲染线程
 - 场景管理与渲染数据管理
 - Deferred Shading Scene Renderer
- 虚幻4的VR渲染
 - 以Google VR HMD插件为例

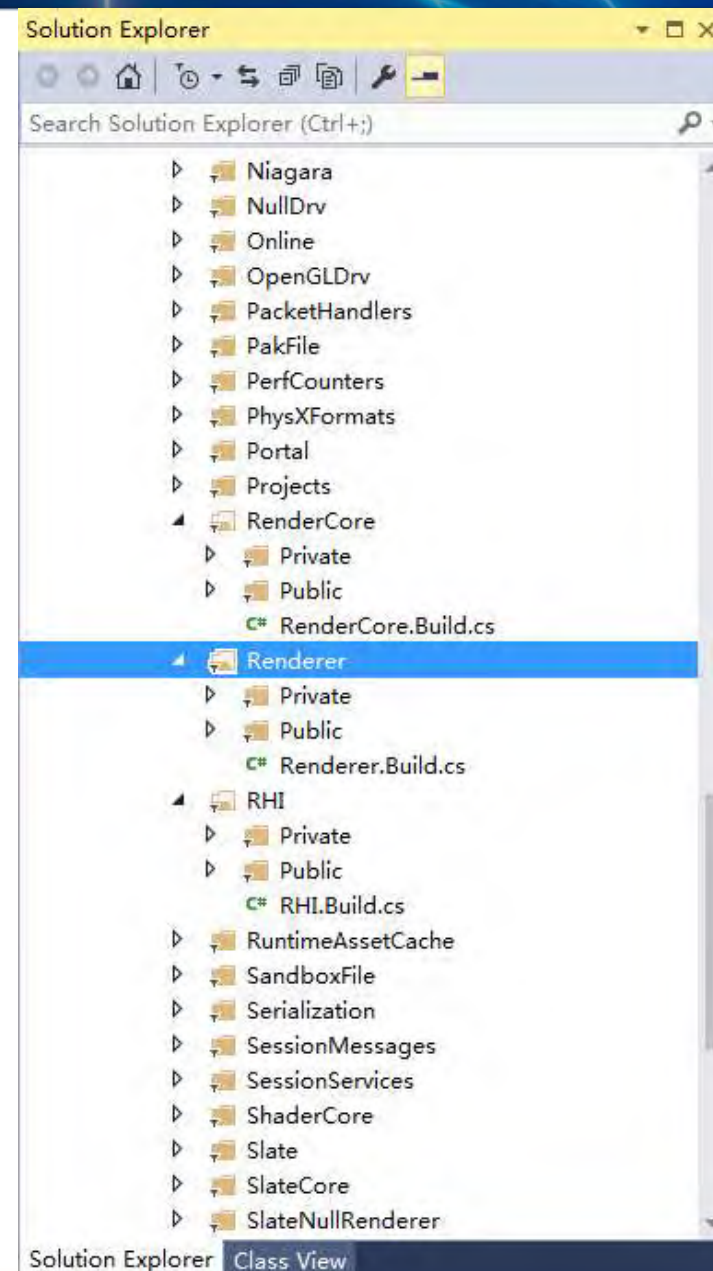


- 渲染系统架构

- 场景管理与渲染器
- 核心问题是管理复杂度和效率



- Engine\Source\Runtime
- 核心代码模块
 - RenderCore
 - Renderer
- RHI抽象层
 - RHI: Render Hardware Interface
 - 基于D3D 11设计
- RHI实现层
 - EmptyRHI
 - Windows\D3D11RHI
 - Apple\MetalRHI
 - OpenGLDrv、VulkanRHI



- 从虚幻3开始引入了渲染线程—Gemini
- 渲染线程对效率的提升
 - 最耗时的系统：渲染、游戏逻辑/脚本、物理模拟
 - 没有渲染线程

Game	Render	Game	Render	Game	Render
Frame 1		Frame 2		Frame 3	

- 使用渲染线程

Game	Game	Game
Render	Render	Render
Frame 1	Frame 2	Frame 3

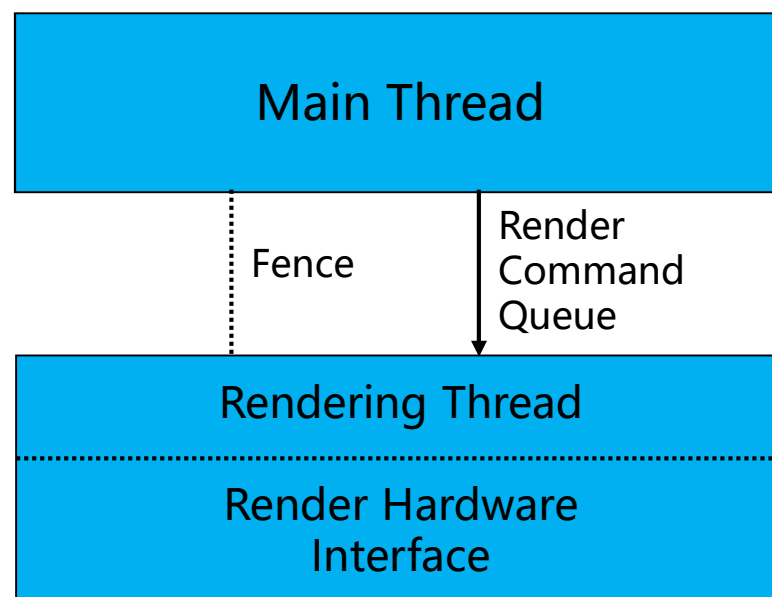


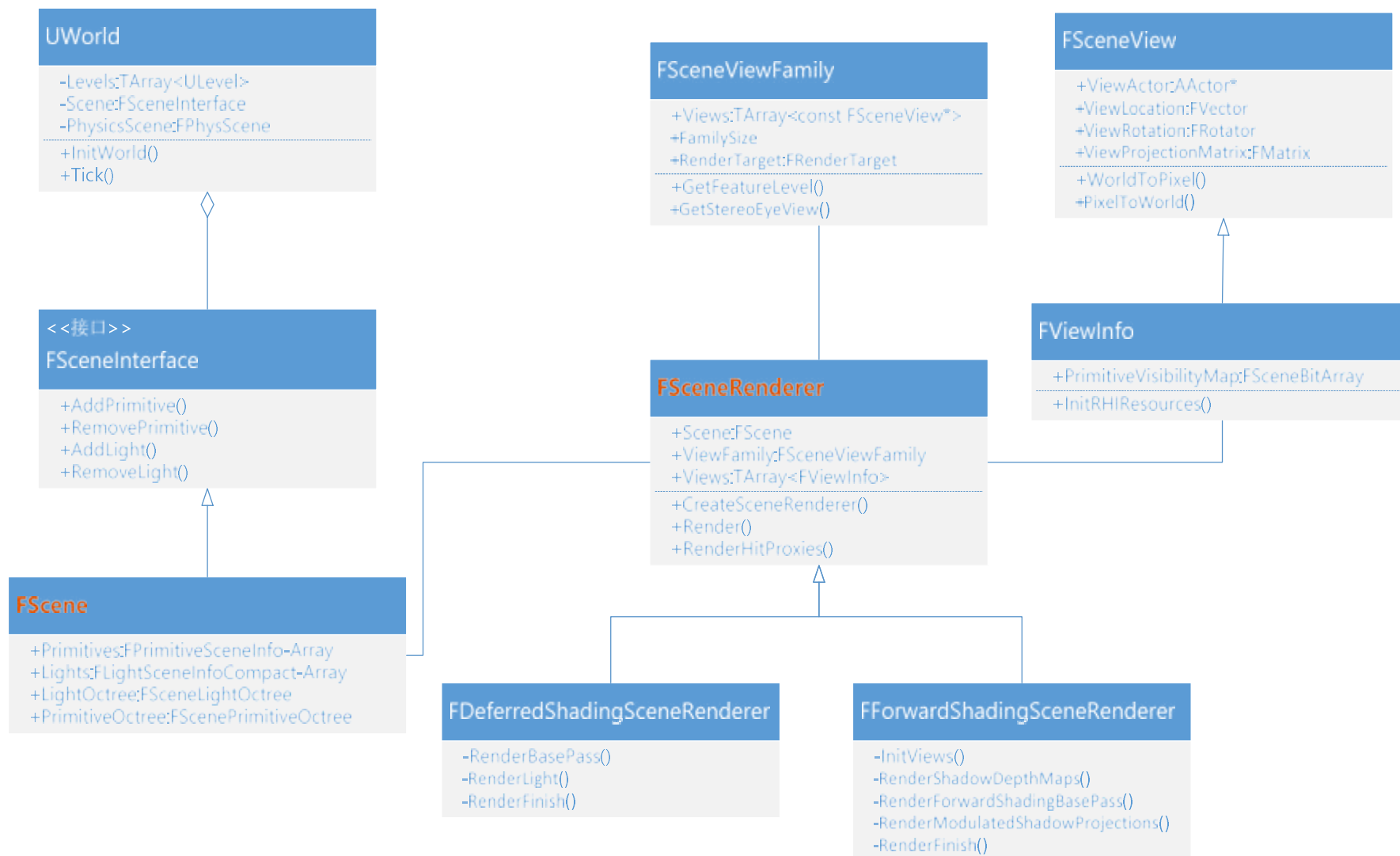
- 主线程领先2帧

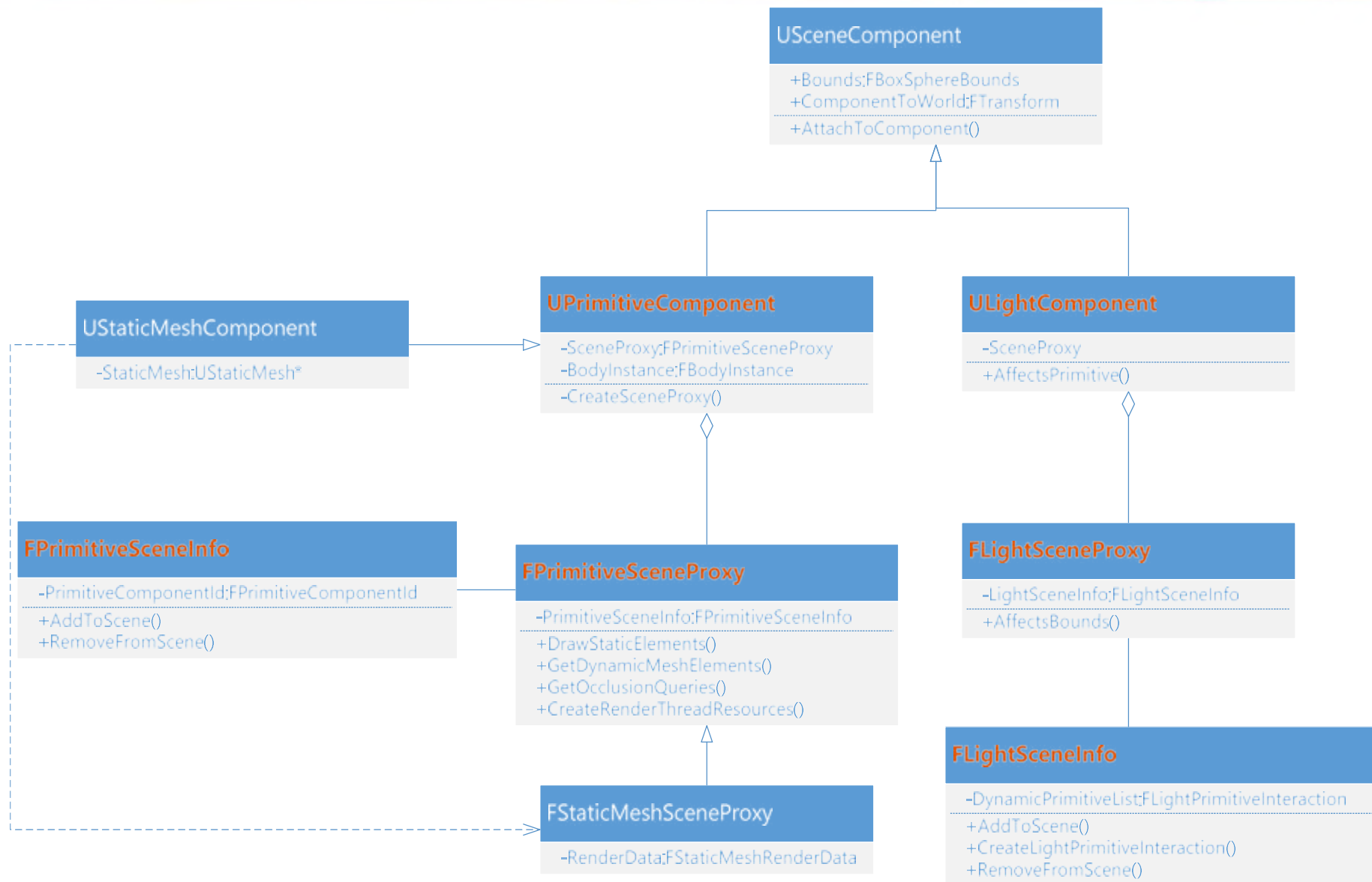
- Front Buffer: N, Render Thread: N+1, Game Thread: N+2
- Render Command Fence: 防止Game Thread跑太快

- 场景数据管理

- 主线程管理场景数据；渲染线程使用Proxy对象；
- 渲染线程管理一份当前帧的拷贝







- Game线程

```
void UGameEngine::Tick( float DeltaSeconds, bool bIdleMode )
{
    UGameEngine::RedrawViewports()
    {
        void FViewport::Draw( bool bShouldPresent)
        {
            void UGameViewportClient::Draw()
            {
                //-- 计算ViewFamily、View的各种属性
                ULocalPlayer::CalcSceneView();

                //-- 发送渲染命令: FDrawSceneCommand
                FRendererModule::BeginRenderingViewFamily()

                //-- Draw HUD
                PlayerController->MyHUD->PostRender();
            }
        }
    }

    FrameEndSync.Sync();
}
```

- Game线程

```
void FRendererModule::BeginRenderingViewFamily()
{
    // render proxies update
    World->SendAllEndOfFrameUpdates();

    // Construct the scene renderer.
    // This copies the view family attributes
    // into its own structures.
    FSceneRenderer* SceneRenderer =
        FSceneRenderer::CreateSceneRenderer(ViewFamily);

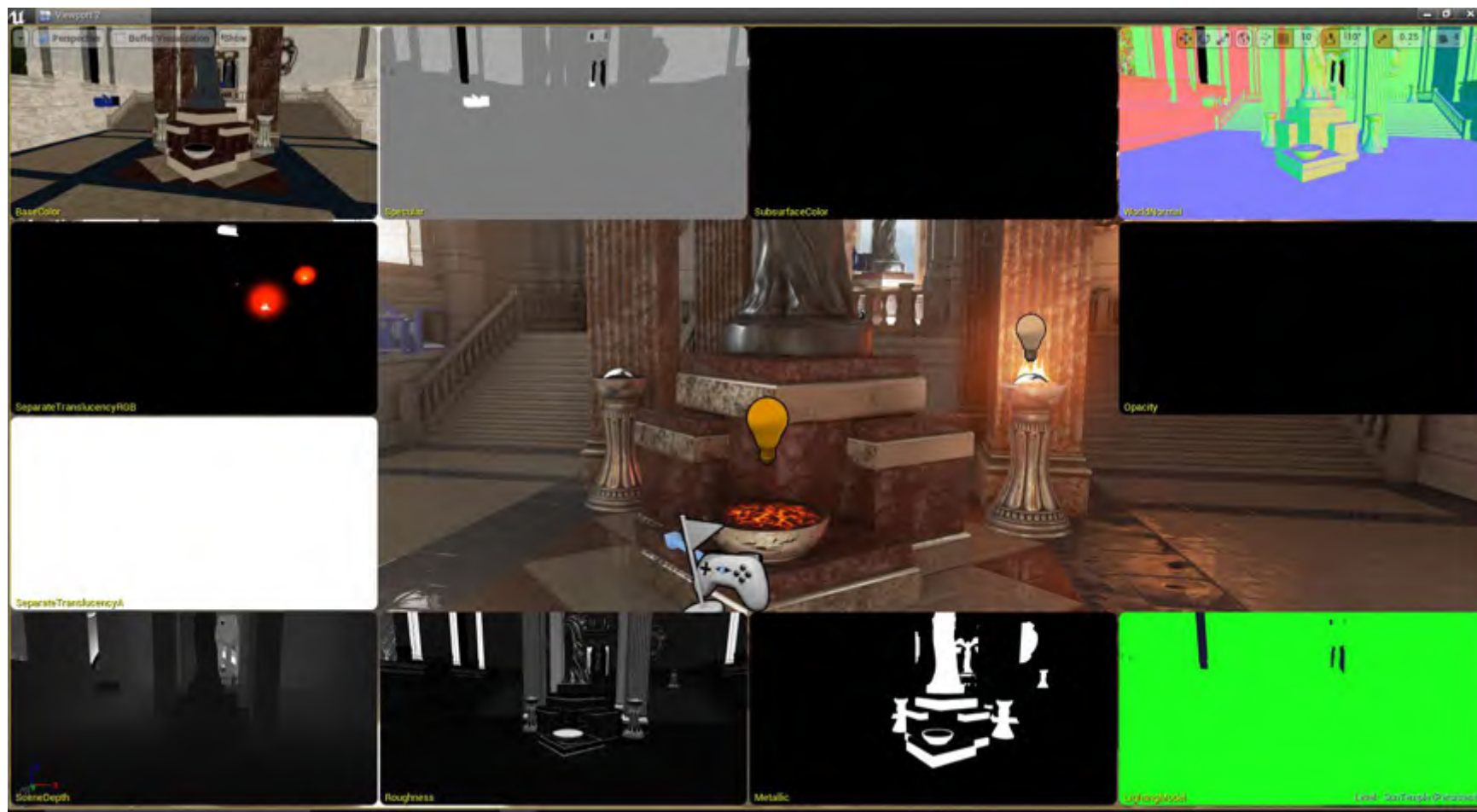
    ENQUEUE_UNIQUE_RENDER_COMMAND_ONEPARAMETER(
        FDrawSceneCommand,
        FSceneRenderer*, SceneRenderer, SceneRenderer,
        {
            RenderViewFamily_RenderThread(RHICmdList, SceneRenderer);
            FlushPendingDeleteRHResources_RenderThread();
        });
}
```

- 渲染线程
 - RenderViewFamily_RenderThread()
 - FSceneRenderer::Render()
 - InitViews()
 - Primitive Visibility Determination: Frustum Cull+Occlusion Cull
 - Light Visibility: Frustum Cull
 - 透明物体排序: Back to Front
 - 不透明物体排序: Front to Back
 - 渲染Scene Color
 - Post Process

- FDeferredShadingSceneRenderer
 - GBuffers: class FSceneRenderTargets
 - DeferredShadingCommon.usf

Name	Format	Usage
GBufferA	PF_A2B10G10R10	WorldNormal: xyz
GBufferB	PF_B8G8R8A8	Metallic: r Specular: g Roughness: b ShadingModelID: a
GBufferC	PF_B8G8R8A8	BaseColor: rgb GBufferAO: a

- FDeferredShadingSceneRenderer



```
void FDeferredShadingSceneRenderer::Render()
{
    bool FDeferredShadingSceneRenderer::InitViews()
    {
        //-- Visibility determination.
        void FSceneRenderer::ComputeViewVisibility()
        {
            FrustumCull();
            OcclusionCull();
        }

        //-- 透明对象排序: back to front
        FTranslucentPrimSet::SortPrimitives();

        //determine visibility of each light
        DoFrustumCullForLights();

        //-- Base Pass对象排序: front to back
        void FDeferredShadingSceneRenderer::SortBasePassStaticData();
    }

    // 下页继续→
}
```

```
void FDeferredShadingSceneRenderer::Render()
{
    //-- EarlyZPass
    FDeferredShadingSceneRenderer::RenderPrePass();
    RenderOcclusion();

    //-- Build Gbuffers
    SetAndClearViewGBuffer();
    FDeferredShadingSceneRenderer::RenderBasePass();
    FSceneRenderTarget::FinishRenderingGBuffer();

    //-- Lighting stage
    RenderLights();
    RenderDynamicSkyLighting();
    RenderAtmosphere();
    RenderFog();
    RenderTranslucency();
    RenderDistortion();
    //-- post processing
    SceneContext.ResolveSceneColor();
    FPostProcessing::Process();
    FDeferredShadingSceneRenderer::RenderFinish();
}
```

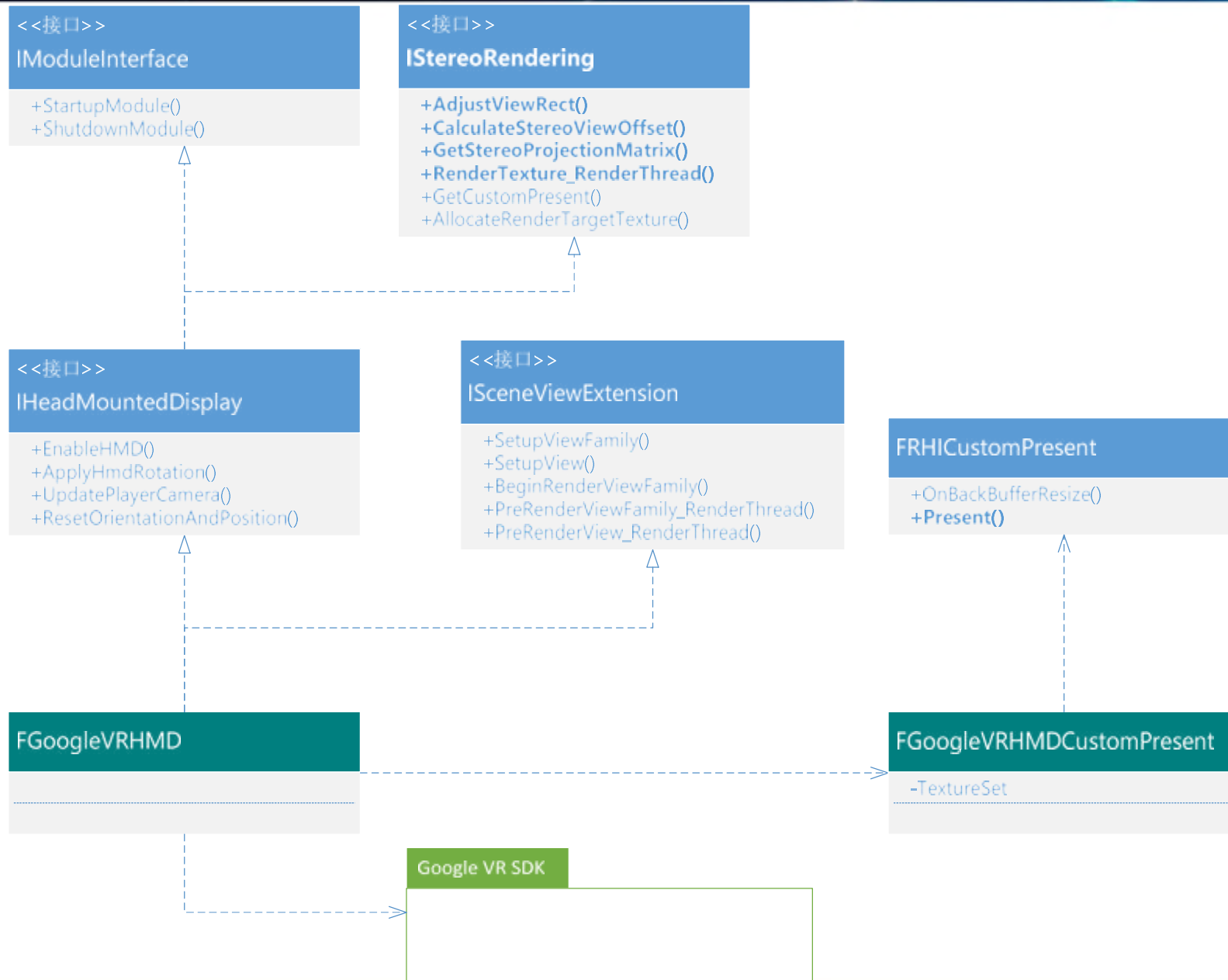


```

void FDeferredShadingSceneRenderer::RenderLights()
{
    foreach(FLightSceneInfoCompact light IN Scene->Lights)
    {
        void FDeferredShadingSceneRenderer::RenderLight(Light)
        {
            RHICmdList.SetBlendState(Additive Blending);
            // DeferredLightVertexShaders.usf
            VertexShader = TDeferredLightVS;
            // DeferredLightPixelShaders.usf
            PixelShader = TDeferredLightPS;
            switch(Light Type)
            {
                case LightType_Directional:
                    DrawFullScreenRectangle();
                case LightType_Point:
                    StencilGeometry::DrawSphere();
                case LightType_Spot:
                    StencilGeometry::DrawCone();
            }
        }
    }
}
    
```

- 虚幻4的渲染系统与Unity3D的架构不同
 - 在Unity3D中，Camera管理一个渲染管线
 - 虚幻4将VR渲染流程整合进引擎底层
 - Scene View Family、Scene View
- Case Study : Google VR HMD插件
 - 代码目录: Plugins/Runtime/GoogleVR/GoogleVRHMD
 - VR渲染系统的静态结构
 - VR渲染流程概述





- 创建HMD Device

//-- 在引擎启动时，会创建所有的HMD设备

```
void UEngine::Init()
{
    bool UEngine::InitializeHMDDDevice()
    {
        for (auto HMDModuleIt = HMDModules.CreateIterator();
             HMDModuleIt; ++HMDModuleIt)
        {
            IHeadMountedDisplayModule* HMDModule = *HMDModuleIt;
            HMDDDevice = HMDModule->CreateHeadMountedDisplay();
        }
    }
}
```

- 绘制流程入口

```
//-- 在View绘制时，如果是Stereo则绘制两个View
void UGameViewportClient::Draw()
{
    const bool bEnableStereo =
        GEngine->IsStereoscopic3D(InViewport);
    int32 NumViews = bEnableStereo ? 2 : 1;

    for (int32 i = 0; i < NumViews; ++i)
    {
    }
}
```

- IStereoRendering接口调用

```
void UGameViewportClient::Draw()  
{  
    ULocalPlayer::CalcSceneView()  
    {  
        ULocalPlayer::GetProjectionData()  
        {  
            GEngine->StereoRenderingDevice->AdjustViewRect(StereoPass);  
            GEngine->StereoRenderingDevice->CalculateStereoViewOffset(StereoPass);  
            ProjectionData.ProjectionMatrix  
                = GEngine->StereoRenderingDevice  
                    ->GetStereoProjectionMatrix(StereoPass);  
        }  
    }  
}
```

- GoogleVRHMD实现

```
void FGoogleVRHMD::AdjustViewRect(StereoPass, int32& X,
                                   int32& Y, uint32& SizeX, uint32& SizeY) const
{
    SizeX = SizeX / 2;
    if( StereoPass == eSSP_RIGHT_EYE )
        X += SizeX;
}
```

```
void FGoogleVRHMD::CalculateStereoViewOffset()
{
    const float EyeOffset = (GetInterpupillaryDistance() * 0.5f)
        * WorldToMeters;
    const float PassOffset = (StereoPassType == eSSP_LEFT_EYE) ?
        -EyeOffset : EyeOffset;
    ViewLocation +=
    ViewRotation.Quaternion().RotateVector(FVector(0, PassOffset, 0
    ));
}
```


- GoogleVRHMD实现

```
void FGoogleVRHMD::RenderTexture_RenderThread()
{
    gvr_distort_to_screen(GVRAPI,
        SrcTexture->GetNativeResource(),
        CachedDistortedRenderTextureParams,
        &CachedPose,
        &CachedFuturePoseTime);
}
```



中国移动开发者大会
Mobile Developer Conference China 2016

THANKS



扫码关注我的微信公众号

mdcc.csdn.net