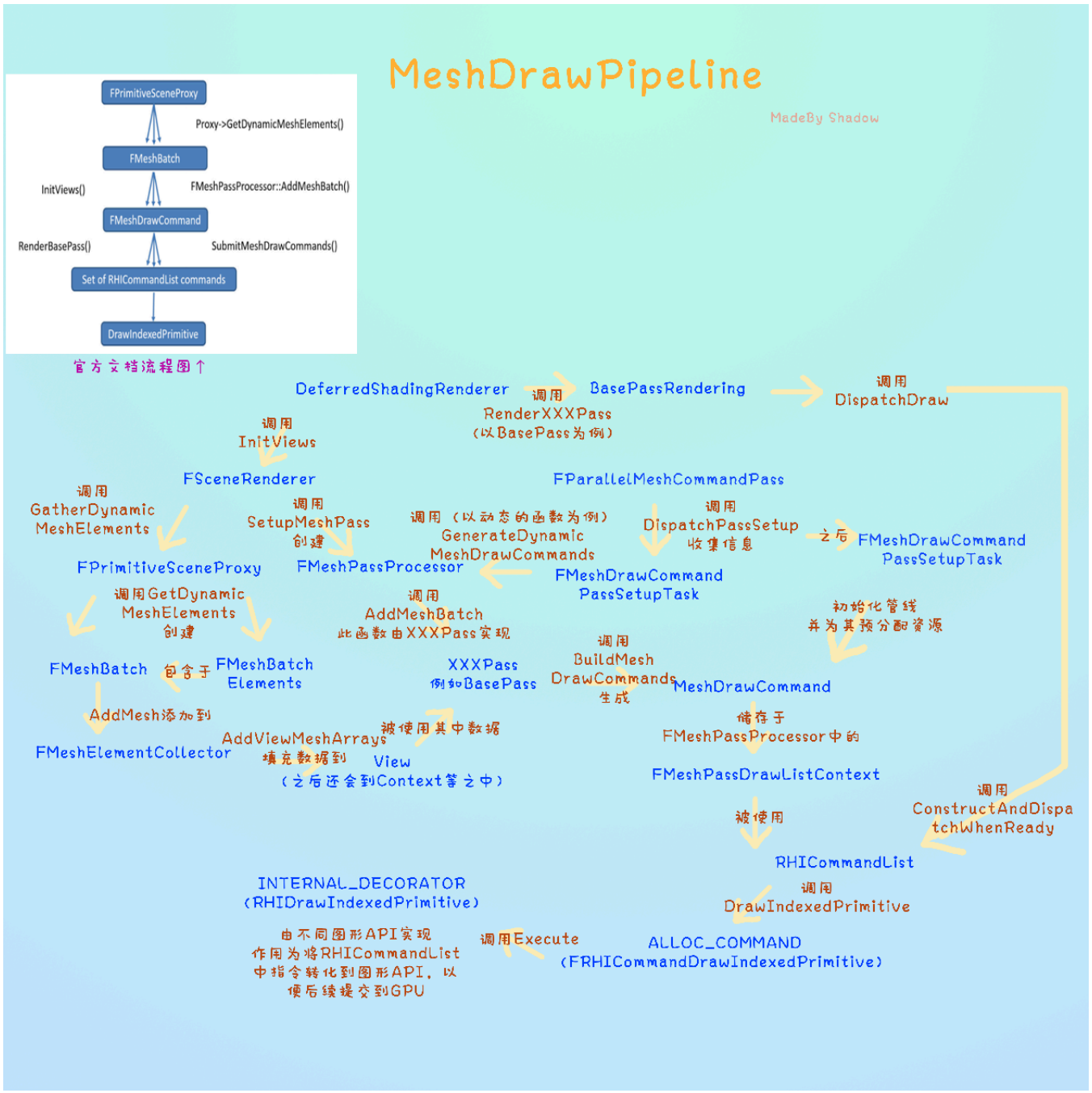
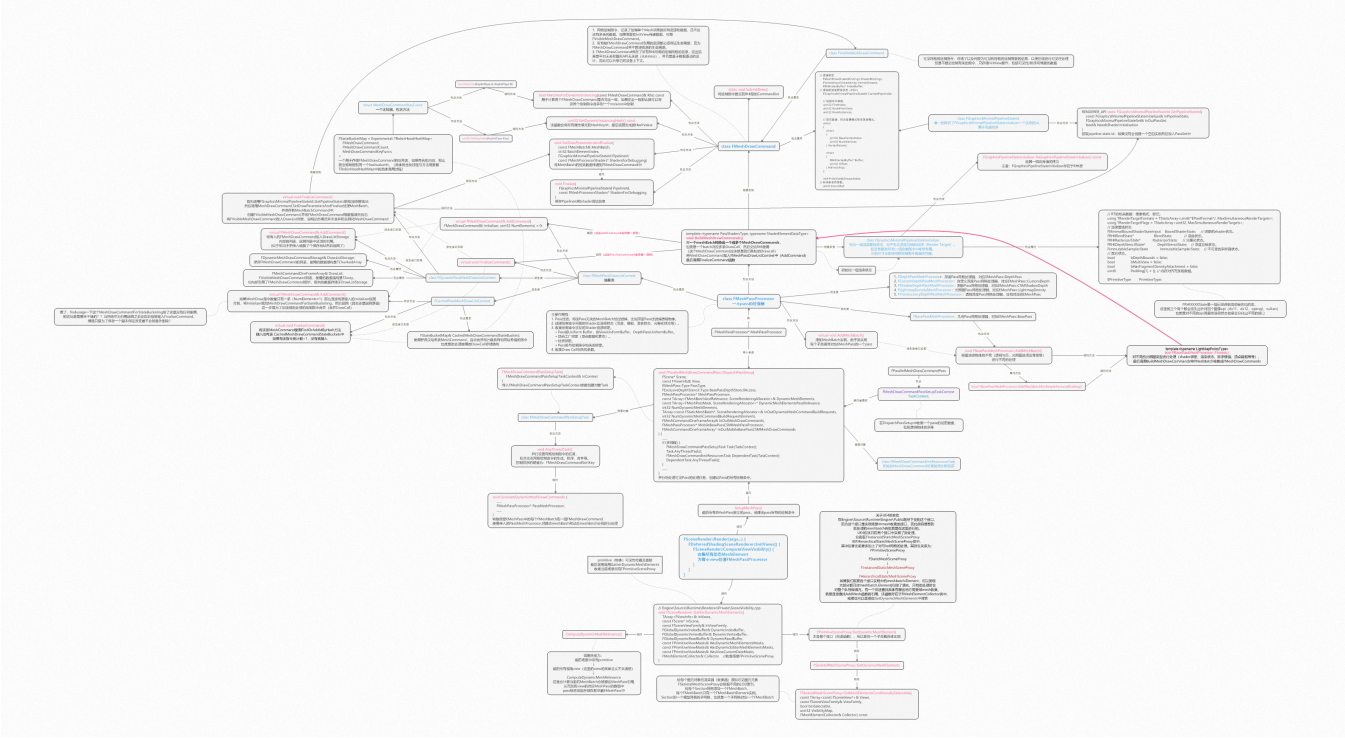
Unreal Mesh Render Pipeline Analysis

Unreal Engine Mesh Draw Pipeline

原文链接 图片链接



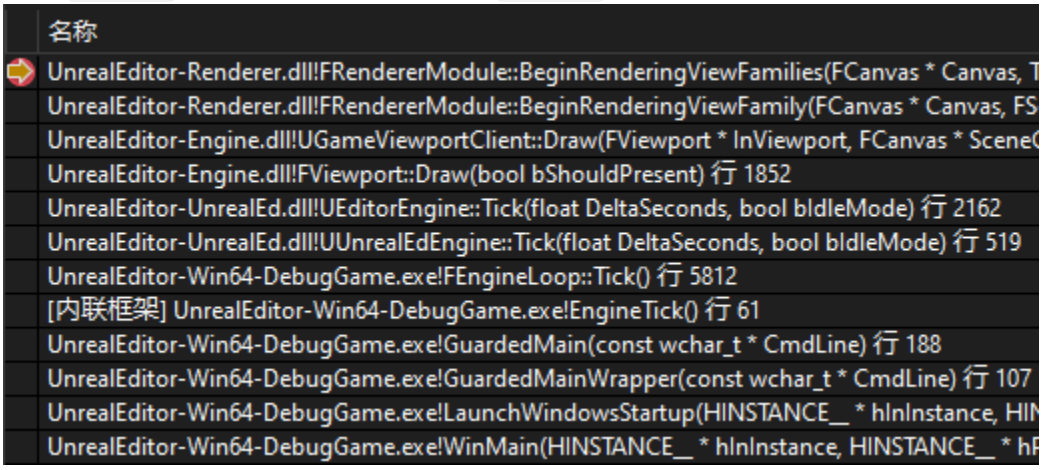
网格处理的类与方法



Render Path Overview

Entry Overview

Unreal Engine 5.2 中的渲染绘制的入口在 `FViewPort::Draw()`，调用堆栈如下图所示，`Draw()` 函数由引擎的更新函数 `Tick()` 每帧调用。



`Draw()` 中会调用 `FRendererModule::BeginRenderingViewFamily`（`FRendererModule::BeginRenderingViewFamilies`），向渲染线程发送消息以开始渲染流程。

```
// Engine\Source\Runtime\Engine\Private\GameViewportClient.cpp
void UGameViewportClient::Draw(FViewport* InViewport, FCanvas* SceneCanvas)
{
...
        GetRendererModule().BeginRenderingViewFamily(SceneCanvas, &ViewFamily);
...
}
```

`FRendererModule::BeginRenderingViewFamilies`中会入队在渲染线程执行的命令`RenderViewFamilies_RenderThread`，该方法会调用`FDeferredShadingSceneRenderer::Render`或`FMobileSceneRenderer::Render`开启场景的绘制。

```
// Engine\Source\Runtime\Renderer\Private\SceneRendering.cpp
void FRendererModule::BeginRenderingViewFamilies(FCanvas* Canvas, TArrayView<FSceneViewFamily*> Vi
{
...
                ENQUEUE_RENDER_COMMAND(FDrawSceneCommand)(
                        [LocalSceneRenderers = CopyTemp(SceneRenderers), DrawSceneEnqueue](FRHICom
                        {
                                uint64 SceneRenderStart = FPlatformTime::Cycles64();
                                const float StartDelayMillisec = FPlatformTime::ToMilliseconds64(S
                                CSV_CUSTOM_STAT_GLOBAL(DrawSceneCommand_StartDelay, StartDelayMill
                                RenderViewFamilies_RenderThread(RHICmdList, LocalSceneRenderers);
                                FlushPendingDeleteRHIResources_RenderThread();
                        });
...
}

static void RenderViewFamilies_RenderThread(FRHICommandListImmediate& RHICmdList, const TArray<FSc
{
...
        // Render the scene.
        SceneRenderer->Render(GraphBuilder);
...
}
```

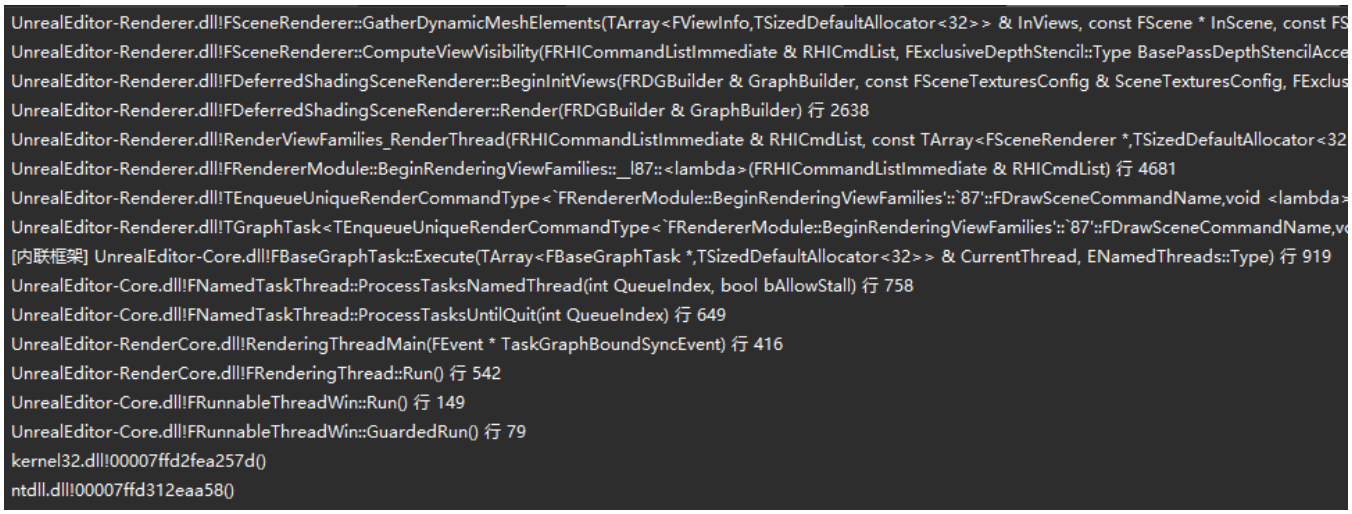
我们主要关注延迟渲染管线，`FDeferredShadingSceneRenderer::Render`中调

用`FDeferredShadingSceneRenderer::BeginInitViews`开始初始化场景的视图、检查可见性、构建可视网格绘制命令等，调用`RenderXXXPass()`以开始不同Pass的渲染。

Call Stack Analysis

下面我们从三个较为重要的 ***CallStack*** 入手进行分析

Call Stack1: Get MeshBatch



`FSceneRenderer::GatherDynamicMeshElements`在执行可见性检查`FSceneRenderer::ComputeViewVisibility`时被调用，其中的`FPrimitiveSceneProxy::GetDynamicMeshElements`是给每个图元对象向渲染器（收集器）添加可见图元元素的机会，由具体的子类实现，如`FLineBatcherSceneProxy`，`FStaticMeshSceneProxy`，`FSkeletalMeshSceneProxy`等。

`Engine\Source\Runtime\Renderer\Private\SceneVisibility.cpp`

```
void FSceneRenderer::GatherDynamicMeshElements(
        TArray<FViewInfo>& InViews,
        const FScene* InScene,
        const FSceneViewFamily& InViewFamily,
        FGlobalDynamicIndexBuffer& DynamicIndexBuffer,
        FGlobalDynamicVertexBuffer& DynamicVertexBuffer,
        FGlobalDynamicReadBuffer& DynamicReadBuffer,
        const FPrimitiveViewMasks& HasDynamicMeshElementsMasks,
        const FPrimitiveViewMasks& HasDynamicEditorMeshElementsMasks,
        FMeshElementCollector& Collector)
{
        ...
        for (int32 PrimitiveIndex = 0; PrimitiveIndex < NumPrimitives; ++PrimitiveIndex)
                {
                        const uint8 ViewMask = HasDynamicMeshElementsMasks[PrimitiveIndex];

                        if (ViewMask != 0)
                        {
                                ...
                                Collector.SetPrimitive(PrimitiveSceneInfo->Proxy, PrimitiveSceneIn

                                PrimitiveSceneInfo->Proxy->GetDynamicMeshElements(InViewFamily.Vie
                                ...
                        }

        }
        ...
}
```

以`FStaticMeshSceneProxy`为例，会根据不同的LOD索引，为每个Section网格添加一个`FMeshBatch`，然后将`FMeshBatch`加入到`FMeshElementCollector`

`Engine\Source\Runtime\Engine\Private\StaticMeshRender.cpp`

```
void FStaticMeshSceneProxy::GetDynamicMeshElements(const TArray<const FSceneView*>& Views, const F
{
...
for (int32 BatchIndex = 0; BatchIndex < NumBatches; BatchIndex++)
{
    // GetWireframeMeshElement will try SetIndexSource at section index 0
    // and GetMeshElement loops over sections, therefore does not have this issue
    if (LODModel.Sections.Num() > 0)
    {
        FMeshBatch &Mesh = Collector.AllocateMesh();

                ...
                Collector.AddMesh(ViewIndex, MeshElement);
                ...

        }
}
...
}
```

Call Stack2: Build MeshDrawCommand

```
UnrealEditor-Renderer.dll!FMeshPassProcessor::BuildMeshDrawCommands<TMeshProcessorShaders<TBasePassVertexShaderPolicyParamType<FUniformLightMapPolicy>,TBasePassPixelShaderPolicyParamType<FUnifo
UnrealEditor-Renderer.dll!FBasePassMeshProcessor::Process<FUniformLightMapPolicy>(const FMeshBatch & MeshBatch, unsigned __int64 BatchElementMask, int StaticMeshId, const FPrimitiveSceneProxy * PrimitiveScen
UnrealEditor-Renderer.dll!FBasePassMeshProcessor::TryAddMeshBatch(const FMeshBatch & MeshBatch, unsigned __int64 BatchElementMask, const FPrimitiveSceneProxy * PrimitiveSceneProxy, int StaticMeshId, const FM
UnrealEditor-Renderer.dll!FBasePassMeshProcessor::AddMeshBatch(const FMeshBatch & MeshBatch, unsigned __int64 BatchElementMask, const FPrimitiveSceneProxy * PrimitiveSceneProxy, int StaticMeshId) 行 1845
UnrealEditor-Renderer.dll!GenerateDynamicMeshDrawCommands(const FViewInfo & View, EShadingPath ShadingPath, EMeshPass::Type PassType, FMeshPassProcessor * PassMeshProcessor, const TArray<FMeshBatchA
UnrealEditor-Renderer.dll!FMeshDrawCommandPassSetupTask::AnyThreadTask() 行 918
UnrealEditor-Renderer.dll!FParallelMeshDrawCommandPass::DispatchPassSetup(FScene * Scene, const FViewInfo & View, FInstanceCullingContext && InstanceCullingContext, EMeshPass::Type PassType, FExclusiveDepthS
UnrealEditor-Renderer.dll!FSceneRenderer::SetupMeshPass(FViewInfo & View, FExclusiveDepthStencil::Type BasePassDepthStencilAccess, FViewCommands & ViewCommands, FInstanceCullingManager & InstanceCullingM
UnrealEditor-Renderer.dll!FSceneRenderer::ComputeViewVisibility(FRHICommandListImmediate & RHICmdList, FExclusiveDepthStencil::Type BasePassDepthStencilAccess, TArray<FViewCommands,TSizedInlineAllocator<4,
UnrealEditor-Renderer.dll!FDeferredShadingSceneRenderer::BeginInitViews(FRDGBuilder & GraphBuilder, const FSceneTexturesConfig & SceneTexturesConfig, FExclusiveDepthStencil::Type BasePassDepthStencilAccess, FI
UnrealEditor-Renderer.dll!FDeferredShadingSceneRenderer::Render(FRDGBuilder & GraphBuilder) 行 2638
UnrealEditor-Renderer.dll!RenderViewFamilies_RenderThread(FRHICommandListImmediate & RHICmdList, const TArray<FSceneRenderer *,TSizedDefaultAllocator<32>> & SceneRenderers) 行 4413
UnrealEditor-Renderer.dll!FRendererModule::BeginRenderingViewFamilies::__l87::<lambda>(FRHICommandListImmediate & RHICmdList) 行 4681
UnrealEditor-Renderer.dll!TEnqueueUniqueRenderCommandType<`FRendererModule::BeginRenderingViewFamilies'::`87'::FDrawSceneCommandName,void <lambda>(FRHICommandListImmediate &)>::DoTask(ENamed
UnrealEditor-Renderer.dll!TGraphTask<TEnqueueUniqueRenderCommandType<`FRendererModule::BeginRenderingViewFamilies'::`87'::FDrawSceneCommandName,void <lambda>(FRHICommandListImmediate &)>>::Ex
[内联框架] UnrealEditor-Core.dll!FBaseGraphTask::Execute(TArray<FBaseGraphTask *,TSizedDefaultAllocator<32>> & CurrentThread, ENamedThreads::Type) 行 919
UnrealEditor-Core.dll!FNamedTaskThread::ProcessTasksNamedThread(int QueueIndex, bool bAllowStall) 行 758
UnrealEditor-Core.dll!FNamedTaskThread::ProcessTasksUntilQuit(int QueueIndex) 行 649
UnrealEditor-RenderCore.dll!RenderingThreadMain(FEvent * TaskGraphBoundSyncEvent) 行 416
UnrealEditor-RenderCore.dll!FRenderingThread::Run() 行 542
UnrealEditor-Core.dll!FRunnableThreadWin::Run() 行 149
UnrealEditor-Core.dll!FRunnableThreadWin::GuardedRun() 行 79
```

`FSceneRenderer::ComputeViewVisibility`中除了`FSceneRenderer::GatherDynamicMeshElements`外还有一个关键的函数`FSceneRenderer::SetupMeshPass`，这个函数会遍历所有Pass的类型，创建对应的`FMeshPassProcessor`，再调用`FParallelMeshDrawCommandPass::DispatchPassSetup`开始该Pass的创建与MeshDrawCommand的构建。

```
// Engine\Source\Runtime\Renderer\Private\SceneRendering.cpp
void FSceneRenderer::SetupMeshPass(FViewInfo& View, FExclusiveDepthStencil::Type BasePassDepthSten
{
...
                        FMeshPassProcessor* MeshPassProcessor = FPassProcessorManager::CreateMeshP

                        FParallelMeshDrawCommandPass& Pass = View.ParallelMeshDrawCommandPasses[Pa
...
                        Pass.DispatchPassSetup(
                                Scene,
                                View,
                                FInstanceCullingContext(FeatureLevel, &InstanceCullingManager, Vie
                                PassType,
                                BasePassDepthStencilAccess,
                                MeshPassProcessor,
                                View.DynamicMeshElements,
                                &View.DynamicMeshElementsPassRelevance,
                                View.NumVisibleDynamicMeshElements[PassType],
                                ViewCommands.DynamicMeshCommandBuildRequests[PassType],
                                ViewCommands.NumDynamicMeshCommandBuildRequestElements[PassType],
                                ViewCommands.MeshCommands[PassIndex]);
...
}
```

`FParallelMeshDrawCommandPass::DispatchPassSetup`中先收集Pass相关的信息到TaskContext中，再调用`FMeshDrawCommandPassSetupTask::AnyThreadTask`创建任务和`FMeshDrawCommandInitResourcesTask::AnyThreadTask`初始化资源。

```
// Engine\Source\Runtime\Renderer\Private\MeshDrawCommands.cpp
void FParallelMeshDrawCommandPass::DispatchPassSetup(
        FScene* Scene,
        const FViewInfo& View,
        FInstanceCullingContext&& InstanceCullingContext,
        EMeshPass::Type PassType,
        FExclusiveDepthStencil::Type BasePassDepthStencilAccess,
        FMeshPassProcessor* MeshPassProcessor,
        const TArray<FMeshBatchAndRelevance, SceneRenderingAllocator>& DynamicMeshElements,
        const TArray<FMeshPassMask, SceneRenderingAllocator>* DynamicMeshElementsPassRelevance,
        int32 NumDynamicMeshElements,
        TArray<const FStaticMeshBatch*, SceneRenderingAllocator>& InOutDynamicMeshCommandBuildRequ
        int32 NumDynamicMeshCommandBuildRequestElements,
        FMeshCommandOneFrameArray& InOutMeshDrawCommands,
        FMeshPassProcessor* MobileBasePassCSMMeshPassProcessor,
        FMeshCommandOneFrameArray* InOutMobileBasePassCSMMeshDrawCommands
)
{
...
        if (bExecuteInParallel)
                {
                        if (IsOnDemandShaderCreationEnabled())
                        {
                                TaskEventRef = TGraphTask<FMeshDrawCommandPassSetupTask>::CreateTa
                        }
                        else
                        {
                                FGraphEventArray DependentGraphEvents;
                                DependentGraphEvents.Add(TGraphTask<FMeshDrawCommandPassSetupTask>
                                TaskEventRef = TGraphTask<FMeshDrawCommandInitResourcesTask>::Crea
                        }
                }
                else
                {
                        QUICK_SCOPE_CYCLE_COUNTER(STAT_MeshPassSetupImmediate);
                        FMeshDrawCommandPassSetupTask Task(TaskContext);
                        Task.AnyThreadTask();
                        if (!IsOnDemandShaderCreationEnabled())
                        {
```

```
                                FMeshDrawCommandInitResourcesTask DependentTask(TaskContext);
                                DependentTask.AnyThreadTask();
                        }
                }
...
}
```

`GenerateDynamicMeshDrawCommands`将会转换指定`EMeshPass`中的每个`FMeshBatch`到一组`FMeshDrawCommand`，其中既会处理动态网格批次`DynamicMeshBatches`也会处理静态网格批次`StaticMeshBatches`，`DynamicMeshCommandBuildRequests`的数量即`NumDynamicMeshCommandBuildRequestElements`，代表`StaticMeshBatches`的数量。`AddMeshBatch`为开始将该`FMeshBatch`转换成`FMeshDrawCommand`的入口。

```
// Engine\Source\Runtime\Renderer\Private\MeshDrawCommands.cpp
void GenerateDynamicMeshDrawCommands(
        const FViewInfo& View,
        EShadingPath ShadingPath,
        EMeshPass::Type PassType,
        FMeshPassProcessor* PassMeshProcessor,
        const TArray<FMeshBatchAndRelevance, SceneRenderingAllocator>& DynamicMeshElements,
        const TArray<FMeshPassMask, SceneRenderingAllocator>* DynamicMeshElementsPassRelevance,
        int32 MaxNumDynamicMeshElements,
        const TArray<const FStaticMeshBatch*, SceneRenderingAllocator>& DynamicMeshCommandBuildReq
        int32 MaxNumBuildRequestElements,
        FMeshCommandOneFrameArray& VisibleCommands,
        FDynamicMeshDrawCommandStorage& MeshDrawCommandStorage,
        FGraphicsMinimalPipelineStateSet& MinimalPipelineStatePassSet,
        bool& NeedsShaderInitialisation
)
{
        ...
        {
                const int32 NumCommandsBefore = VisibleCommands.Num();
                const int32 NumDynamicMeshBatches = DynamicMeshElements.Num();

                for (int32 MeshIndex = 0; MeshIndex < NumDynamicMeshBatches; MeshIndex++)
                {
                        if (!DynamicMeshElementsPassRelevance || (*DynamicMeshElementsPassRelevanc
                        {
                                const FMeshBatchAndRelevance& MeshAndRelevance = DynamicMeshElemen
                                const uint64 BatchElementMask = ~0ull;

                                PassMeshProcessor->AddMeshBatch(*MeshAndRelevance.Mesh, BatchEleme
                        }
                }

                const int32 NumCommandsGenerated = VisibleCommands.Num() - NumCommandsBefore;
                checkf(NumCommandsGenerated <= MaxNumDynamicMeshElements,
                        TEXT("Generated %d mesh draw commands for DynamicMeshElements, while preal
        }
        ...
        {
```

```
                const int32 NumCommandsBefore = VisibleCommands.Num();
                const int32 NumStaticMeshBatches = DynamicMeshCommandBuildRequests.Num();

                for (int32 MeshIndex = 0; MeshIndex < NumStaticMeshBatches; MeshIndex++)
                {
                        const FStaticMeshBatch* StaticMeshBatch = DynamicMeshCommandBuildRequests[
                        const uint64 DefaultBatchElementMask = ~0ul;
                        PassMeshProcessor->AddMeshBatch(*StaticMeshBatch, DefaultBatchElementMask,
                }

                const int32 NumCommandsGenerated = VisibleCommands.Num() - NumCommandsBefore;
                checkf(NumCommandsGenerated <= MaxNumBuildRequestElements,
                        TEXT("Generated %d mesh draw commands for DynamicMeshCommandBuildRequests,
        }
        ...
}
```

`FMeshPassProcessor::BuildMeshDrawCommands`中会创建`FMeshDrawCommand`并填充信息，然后将其添加到`DrawListContext`中，最后再执行`FinalizeCommand`完成MeshDrawCommands的构建。

以`FDynamicPassMeshDrawListContext`为例，`FDynamicPassMeshDrawListContext::FinalizeCommand`中使用传入的`FMeshDrawCommand`创建`FVisibleMeshDrawCommand`并添加到`FMeshProcessor`的`DrawList`中，`DrawList`的类型是`FMeshCommandOneFrameArray`，`FMeshCommandOneFrameArray`的定义如下：

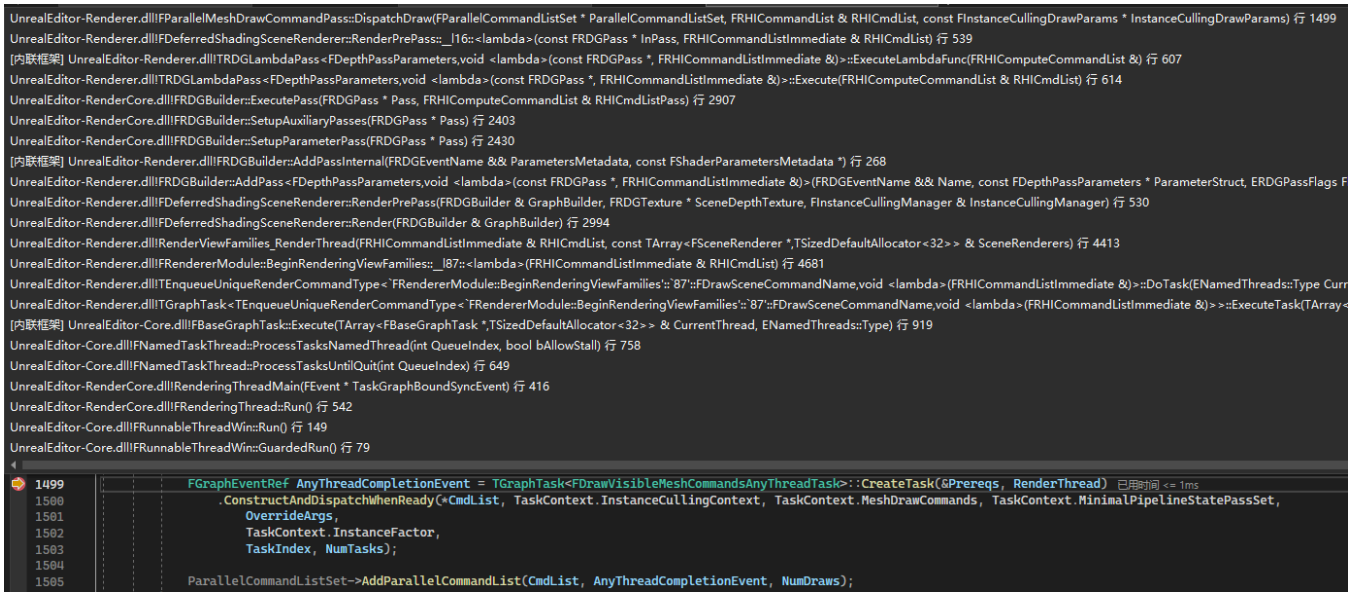
`typedef TArray<FVisibleMeshDrawCommand, SceneRenderingAllocator> FMeshCommandOneFrameArray;`

```
// Engine\Source\Runtime\Renderer\Public\MeshPassProcessor.h
        virtual void FinalizeCommand(
                const FMeshBatch& MeshBatch,
                int32 BatchElementIndex,
                const FMeshDrawCommandPrimitiveIdInfo &IdInfo,
                ERasterizerFillMode MeshFillMode,
                ERasterizerCullMode MeshCullMode,
                FMeshDrawCommandSortKey SortKey,
                EFVisibleMeshDrawCommandFlags Flags,
                const FGraphicsMinimalPipelineStateInitializer& PipelineState,
                const FMeshProcessorShaders* ShadersForDebugging,
                FMeshDrawCommand& MeshDrawCommand) override final
        {
                FGraphicsMinimalPipelineStateId PipelineId = FGraphicsMinimalPipelineStateId::GetP

                MeshDrawCommand.SetDrawParametersAndFinalize(MeshBatch, BatchElementIndex, Pipelin

                FVisibleMeshDrawCommand NewVisibleMeshDrawCommand;
                //@todo MeshCommandPipeline - assign usable state ID for dynamic path draws
                // Currently dynamic path draws will not get dynamic instancing, but they will be
                const FMeshBatchElement& MeshBatchElement = MeshBatch.Elements[BatchElementIndex];
                NewVisibleMeshDrawCommand.Setup(&MeshDrawCommand, IdInfo, -1, MeshFillMode, MeshCu
                        MeshBatchElement.bIsInstanceRuns ? MeshBatchElement.InstanceRuns : nullptr
                        MeshBatchElement.bIsInstanceRuns ? MeshBatchElement.NumInstances : 0
                        );
                DrawList.Add(NewVisibleMeshDrawCommand);
        }
```

Call Stack3: Submit RHICommand



在可见性测试后便是渲染的环节，`FDeferredShadingSceneRenderer`为不同的pass实现了不同的渲染函数并命名为`RenderXXXPass`(XXX:Pass Name)，执行绘制的入口是该pass对应的`ParallelMeshDrawCommandPasses`的`DispatchDraw`方法，以prePass为例，在`RenderPrePass`函数中，绘制的入口以及pass参数先被加入RDG进行资源的处理，再异步地（由RDG决定顺序）以Lambda函数方式被调用。

```
// DepthRendering.cpp
void FDeferredShadingSceneRenderer::RenderPrePass(FRDGBuilder& GraphBuilder, FRDGTextureRef SceneD
{
...
                                        GraphBuilder.AddPass(
                                                RDG_EVENT_NAME("DepthPassParallel"),
                                                PassParameters,
                                                ERDGPassFlags::Raster | ERDGPassFlags::SkipRenderP
                                                [this, &View, PassParameters](const FRDGPass* InPa
                                        {
                                                FRDGParallelCommandListSet ParallelCommandListSet(
                                                ParallelCommandListSet.SetHighPriority();

                                                View.ParallelMeshDrawCommandPasses[EMeshPass::Dept
                                        });
...
}
```

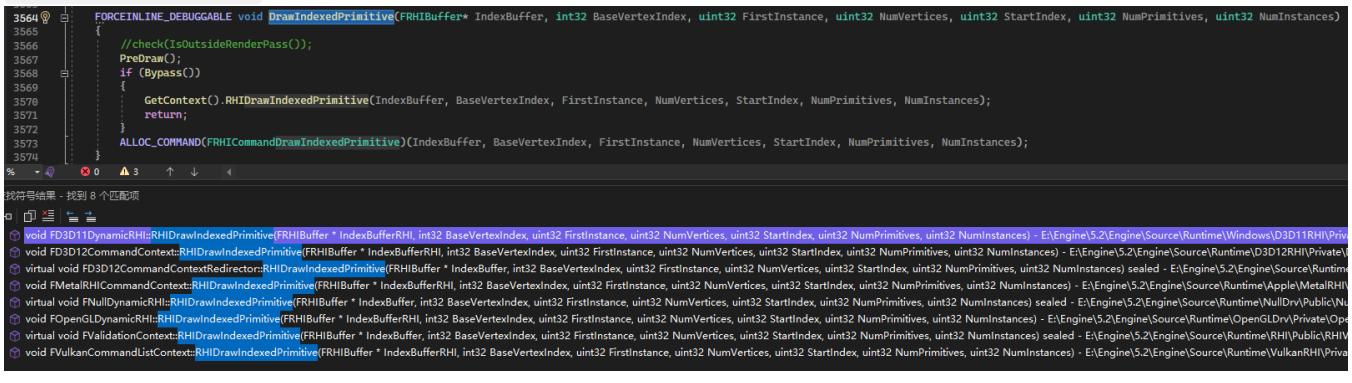
`FDrawVisibleMeshCommandAnyThreadTask`执行的绘制指令单个MeshDrawCommand的绘制命令的创建的调用堆栈如下，绘制的任务并不是立即执行的，而是由RDG的`Execute()`发起执行。

```
UnrealEditor-Renderer.dll!FMeshDrawCommand::SubmitDrawEnd(const FMeshDrawCommand & MeshDrawCommand, unsig
UnrealEditor-Renderer.dll!FMeshDrawCommand::SubmitDraw(const FMeshDrawCommand & MeshDrawCommand, const Exp
UnrealEditor-Renderer.dll!FInstanceCullingContext::SubmitDrawCommands(const TArray<FVisibleMeshDrawCommand,TConc
[内联框架] UnrealEditor-Renderer.dll!FDrawVisibleMeshCommandsAnyThreadTask::DoTask(ENamedThreads::Type) 行 1403
```

```
// MeshPassProcessor.cpp
void FMeshDrawCommand::SubmitDrawEnd(const FMeshDrawCommand& MeshDrawCommand, uint32 InstanceFacto
        FRHIBuffer* IndirectArgsOverrideBuffer,
        uint32 IndirectArgsOverrideByteOffset)
{
    RHICmdList.DrawIndexedPrimitive(
                                MeshDrawCommand.IndexBuffer,
                                MeshDrawCommand.VertexParams.BaseVertexIndex,
                                0,
                                MeshDrawCommand.VertexParams.NumVertices,
                                MeshDrawCommand.FirstIndex,
                                MeshDrawCommand.NumPrimitives,
                                MeshDrawCommand.NumInstances * InstanceFactor
                        );
}
```

`DrawIndexedPrimitive`在ByPass情况下会直接调用的方法`RHIDrawIndexedPrimitive`由不同图形驱动的RHI实现，实测在windows下只有在FD3D12对应的RHI中打断点才会生效，并没有使用到其他图形驱动的RHI。

`RHICommandList.h`



以OpenGL为例，该方法会直接调用OpenGL的绘制方法

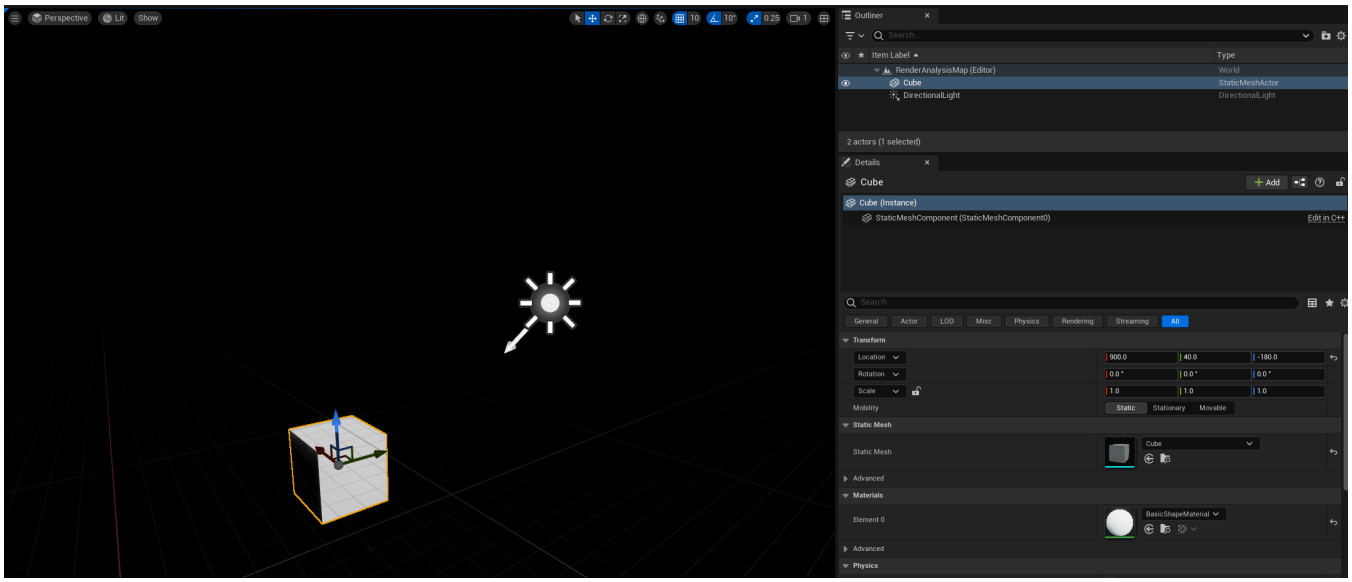
如`glDrawElementsInstanced`和`glDrawElements`等进行绘制。

```
// OpenGLCommands.cpp
void FOpenGLDynamicRHI::RHIDrawIndexedPrimitive(FRHIBuffer* IndexBufferRHI, int32 BaseVertexIndex,
{
    ...
    FOpenGL::DrawElementsInstanced(DrawMode, NumElements, IndexType, INDEX_TO_VOID(StartIndex), Nu
    /*
    OpenGL3.h
    static FORCEINLINE void DrawElementsInstanced(GLenum Mode, GLsizei Count, GLenum Type, const G
        {
                glDrawElementsInstanced(Mode, Count, Type, Indices, InstanceCount);
        }
    */
    ...
    glDrawElements(DrawMode, NumElements, IndexType, INDEX_TO_VOID(StartIndex));
    ...
}
```

Example: PrePass (Depth Pass)

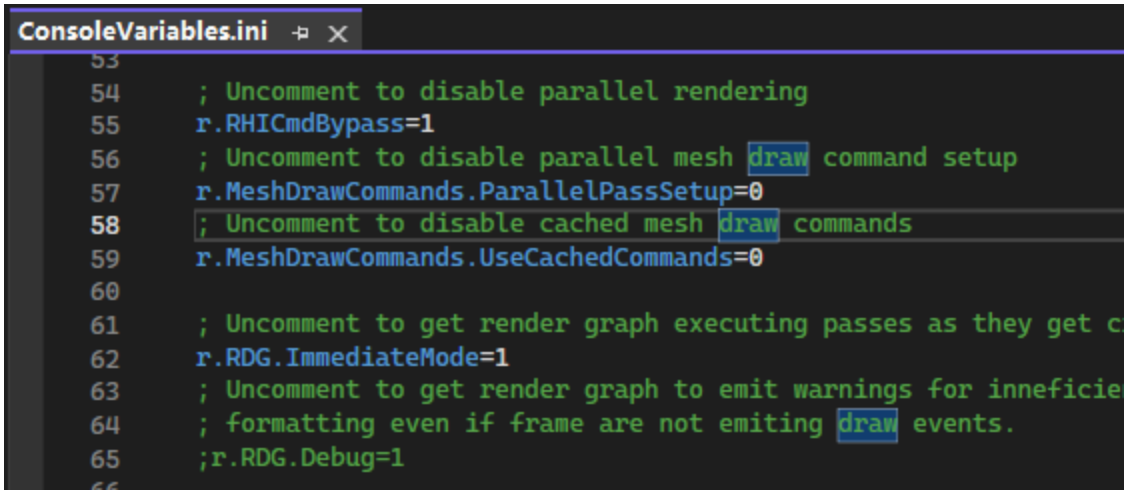
Scene Overview

1 Cube + 1 DirectionalLight



CVars (Console Variables) at Engine\Config\ ConsoleVariables.ini

虚幻的绘制数据分两类，一类是Dynamic一类是Static，不同的绘制数据的绘制路径是不一样的。Static数据是预生成的，将被保存在CachedMeshDrawCommandStateBuckets里，Dynamic的MeshDrawCommand是每帧重新生成的。按照下图设置cvars以后每帧都会重新生成MeshDrawCommand，方便分析。



Cube相关信息截帧

场景信息

游戏线程在`FRendererModule::BeginRenderingViewFamily`中初始化渲染线程的场景渲染器`FSceneRenderer`，然后在渲染线程执行的方法`RenderViewFamilies_RenderThread`中调用`FSceneRenderer`的`Render()`方法进行渲染。
在渲染线程遍历`SceneRenders`执行实际的渲染函数`Render()`前打断点截帧，查看`Scene`里的`Primitives`信息，`Primitives`里的元素实际上是`FPrimitiveSceneInfo`。
`SceneRendering.cpp`

```cpp
/**
 * Helper function performing actual work in render thread.
 *
 * @param SceneRenderers    List of scene renderers to use for rendering.
 */
static void RenderViewFamilies_RenderThread(FRHICommandListImmediate& RHICmdList, const TArray<FSceneRenderer*>& SceneRenderers)
{
    LLM_SCOPE(ELLMTag::SceneRender);

    // All renderers point to the same Scene (calling code asserts this)
    FScene* const Scene = SceneRenderers[0]->Scene;

    FSceneRenderer::RenderThreadBegin(RHICmdList, SceneRenderers);

    bool bAnyShowHitProxies = false;

#if WITH_DEBUG_VIEW_MODES
    // Flag so we only call FGPUSkinCacheVisualizationData::Update and draw the visualization text once
    bool bUpdatedGPUSkinCacheVisualization = false;
#endif

    // update any resources that needed a deferred update
    FDeferredUpdateResource::UpdateResources(RHICmdList);

    for (FSceneRenderer* SceneRenderer : SceneRenderers) 已用时间 <= 10ms
    {
        const ERHIFeatureLevel::Type FeatureLevel = SceneRenderer->FeatureLevel;

        FSceneViewFamily& ViewFamily = SceneRenderer->ViewFamily;

        FRDGBuilder GraphBuilder(
            RHICmdList,
            RDG_EVENT_NAME("SceneRenderer_%s(ViewFamily=%s)",
                ViewFamily.EngineShowFlags.HitProxies ? TEXT("RenderHitProxies") : TEXT("Render"),
                ViewFamily.bResolveScene ? TEXT("Primary") : TEXT("Auxiliary")
            ),
            FSceneRenderer::GetRDGParalelExecuteFlags(FeatureLevel)
        );
```

场景中共有五个`Primitives`。

Primitives	Num=5	TArray<FPrimitiveSceneInfo *,TSizedDef...
[0]	0x0000093830bf2a00 {Proxy=0x00000938471af000 {RenderData=0x00000937e18e9b00 {LODResources=Num=1 LODVertexFactories=...} ...	FPrimitiveSceneInfo *
[1]	0x0000093830bfb400 {Proxy=0x0000093847517800 {RenderData=0x0000093808d36e00 {LODResources=Num=1 LODVertexFactories=...} ...	FPrimitiveSceneInfo *
[2]	0x0000093830bf5100 {Proxy=0x000009383d24f000 {Lines=Empty Points=Empty Meshes=Empty } PrimitiveComponentId=...}	FPrimitiveSceneInfo *
[3]	0x0000093830bf0c00 {Proxy=0x000009383d24b400 {Lines=Empty Points=Empty Meshes=Empty } PrimitiveComponentId=...}	FPrimitiveSceneInfo *
[4]	0x0000093830bf7b00 {Proxy=0x000009383d240400 {Lines=Empty Points=Empty Meshes=Empty } PrimitiveComponentId=...}	FPrimitiveSceneInfo *
[原始视图]	{AllocatorInstance={...} ArrayNum=5 ArrayMax=5 }	TArray<FPrimitiveSceneInfo *,TSizedDef...

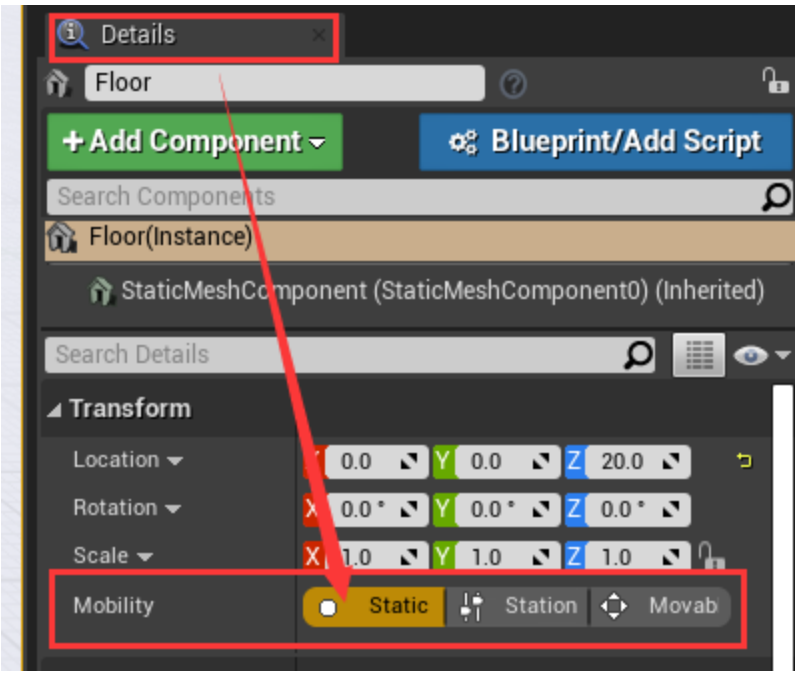
从`Primitives`的`Proxy`的`ResourceName`和`OwnerName`中可以看出第一个primitive对应的是Cube，第二个是Sphere代表的DefaultPawn，剩下三个是LineBatchComponent。

OwnerName	"StaticMeshActor"_3
ResourceName	"Cube"
LevelName	"/Game/UEDPIE_0_RenderAnalysisMap"

OwnerName	"DefaultPawn"_0
ResourceName	"Sphere"

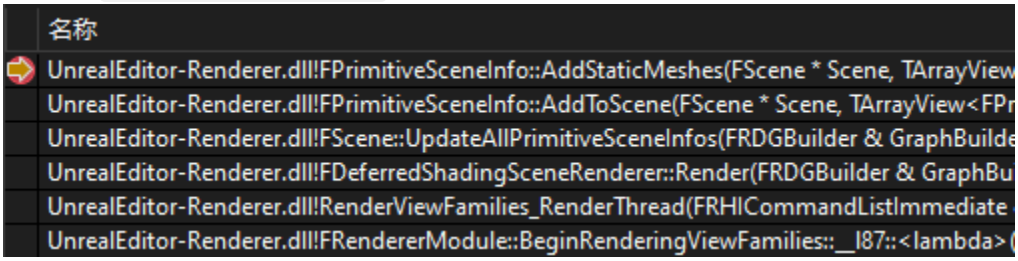
静态绘制路径，缓存MeshBatch

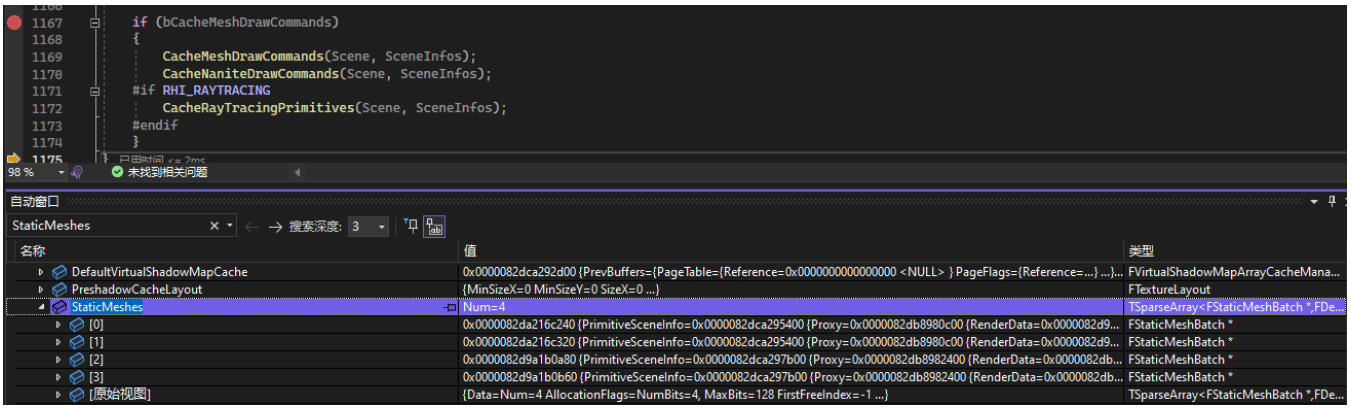
静态绘制路径通常可以被缓存，所以也叫缓存绘制路径，适用的对象可以是静态模型（可在UE编辑器的网格属性面板中指定，见下图）。



静态模型在加入场景后，其对应的`FPrimitiveSceneInfo`在调用`AddStaticMeshes`时，被执行缓存处理，调用堆栈如下所示。`AddStaticMeshes`中会添加静态网格元素到场景的静态网格列表，也会缓存静态的MeshDrawCommand（如果开启了缓存）。

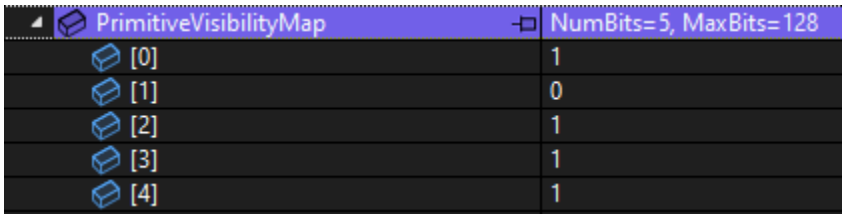
我们添加且只添加一个静态的Cube到场景。打断点可以发现在执行完`AddStaticMeshes`后，场景中的`StaticMeshes`已经有了4个元素，其中第0个和第1个就是我们添加的Cube的MeshBatch，他们的`PrimitiveSceneInfo`是一样的。



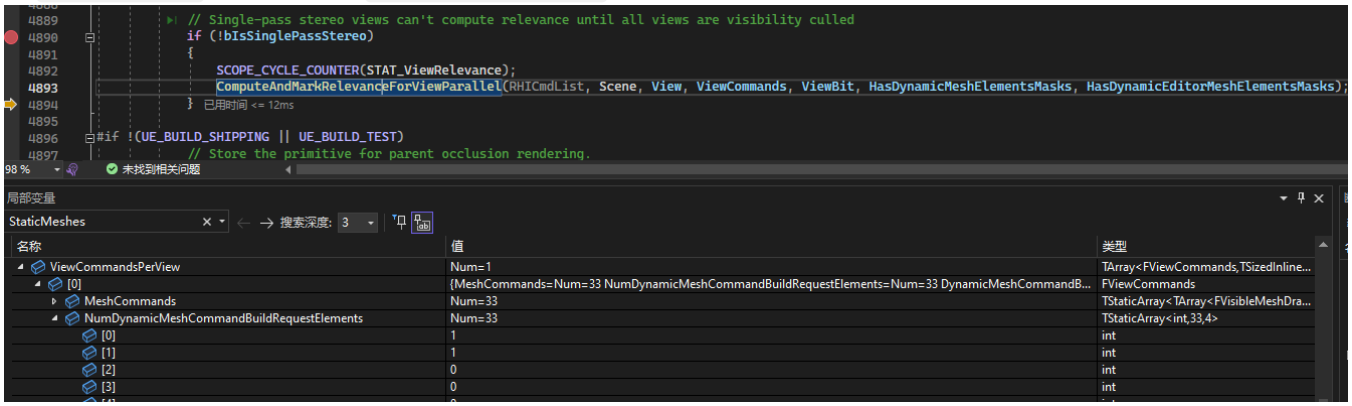


可视性与相关性检测

`SceneVisibility.cpp`里的`FsceneRender::ComputeViewVisibility`中执行各种剔除与可视性检测，然后在`FSceneRenderer::SetupMeshPass`中遍历各个pass生成drawcommand。在构建最后的绘制列表之前需要把不需要的MeshDrawCommand剔除掉，UE有多种剔除算法，可见性剔除，视锥体剔除等。最后会构建一个View.PrimitiveVisibilityMap。这个VisibilityMap会把没用的MeshDrawCommand丢掉，让它无法进入最后的渲染队列里。
在进行剔除与可视性检测后的VisibilityMap中可以看到，Cube的可见性标志位为1（可见），Pawn的Sphere的可见性标志位为0（不可见）。

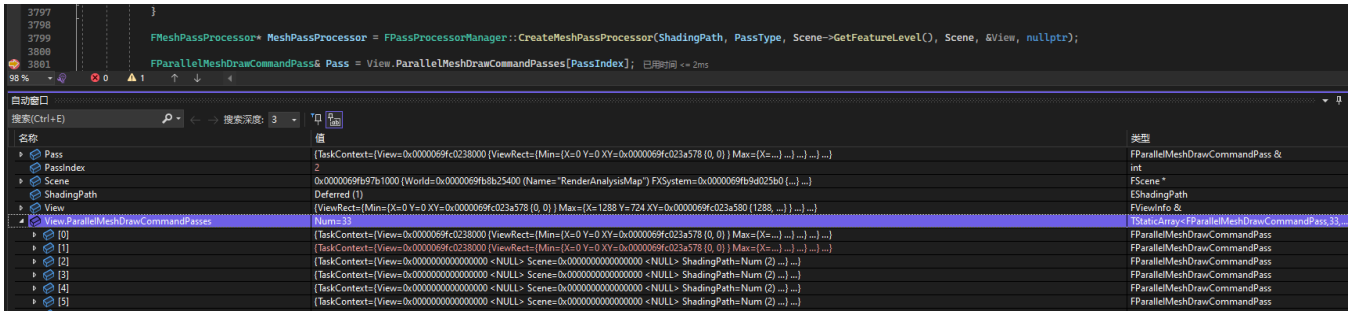


在`FsceneRender::ComputeViewVisibility`中还会进行相关性(`Relevance`)的检测，我们加入场景的`Cube`是静态物体，场景中已经缓存了它的`MeshBatch`，但是由于没有缓存它的`MeshDrawCommand`所以每帧都要重新生成这个静态`MeshBatch`的`MeshDrawCommand`。
在`ComputeAndMarkRelevanceForViewParallel`中会计算相关性并且填充`FViewCommands`中的`NumDynamicMeshCommandBuildRequestElements`等信息，这个对应的就是需要，如下图断电所示，在经过相关性计算后，第一个Pass和第二个Pass对应的`NumDynamicMeshCommandBuildRequestElements`被填充为1，代表该Pass中有一个静态物体生成的`FMeshBatch`需要构建成`MeshDrawCommand`。

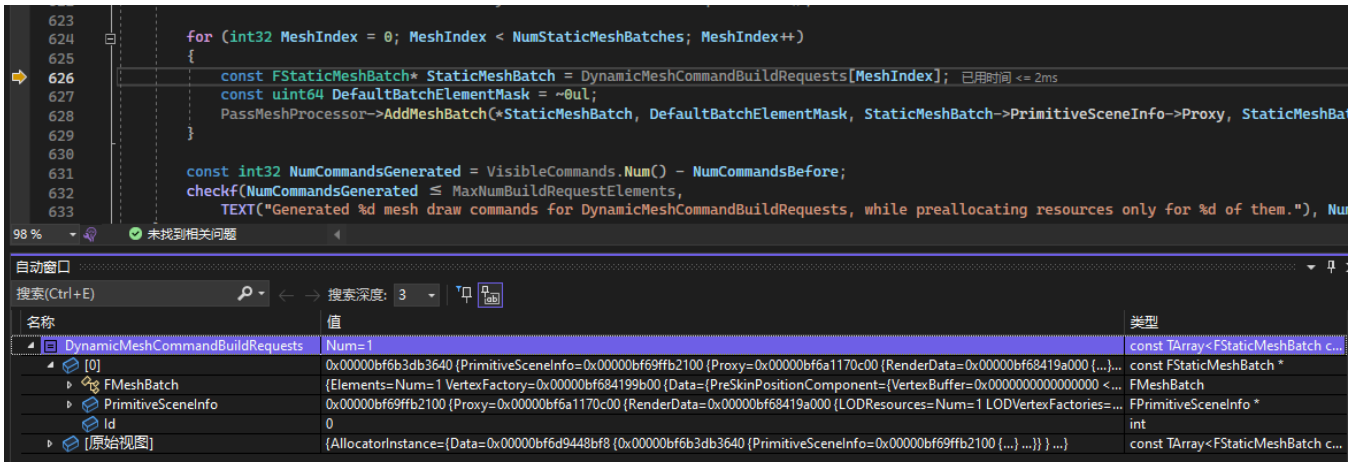


在`FSceneRenderer::SetupMeshPass`中打断点调试可以看到`ParallelMeshDrawCommandPasses`中的各个Pass的信息被逐个填充，其成员变量`TaskContext`里有该Pass对应的MeshDrawCommand信息

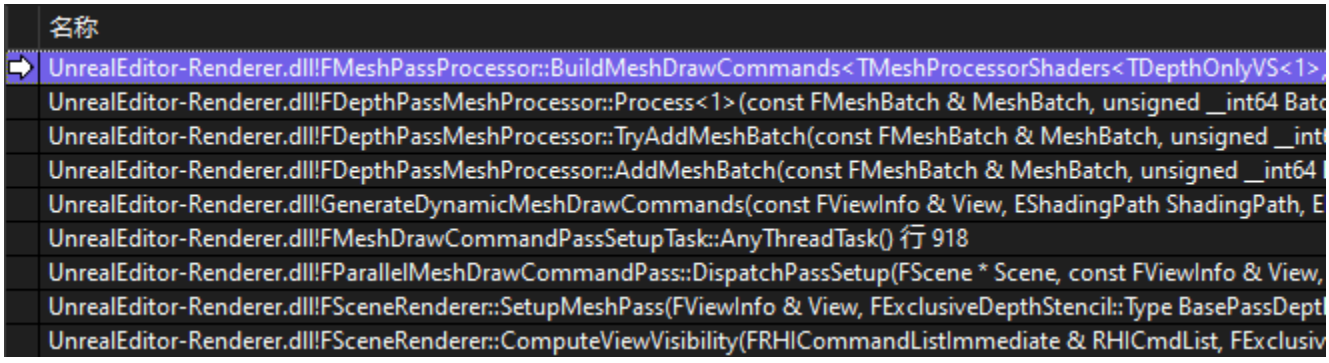
`SceneRendering.cpp`



在`GenerateDynamicMeshDrawCommands`中打断点截帧可以发现，在PrePass构建过程中，没有DynamicMeshBatch，有且只有一个StaticMeshBatch，而且根据`PrimitiveSceneInfo`可以推断这个MeshBatch对应的是Cube。



在`BuildMeshDrawCommand`中打断点截帧进行分析，此时的堆栈如下，可以看到堆栈走的是DepthPass的路径。



在执行`AddCommand`之前，此时的DrawListStorage是空的。

```
144     for (int32 BatchElementIndex = 0; BatchElementIndex < NumElements; BatchElementIndex++)
145     {
146         if ((1ull << BatchElementIndex) & BatchElementMask)
147         {
148             const FMeshBatchElement& BatchElement = MeshBatch.Elements[BatchElementIndex];
149             FMeshDrawCommand& MeshDrawCommand = DrawListContext->AddCommand(SharedMeshDrawCommand, NumElements);  已用时间 <= 10ms
150
```

98 % 未找到相关问题

自动窗口

搜索(Ctrl+E) 搜索深度: 3

名称	值	类型
&BatchElementMask	0x00000051554858c8 {4294967295}	unsigned __int64 *
BatchElement	{PrimitiveUniformBuffer=0x0000000000000000 <NULL> PrimitiveUniformBufferResource=0x0000000000000000 <NULL> ...}	const FMeshBatchElement &
BatchElementIndex	0	int
DrawListContext	0x00000051554861a0 {DrawListStorage={MeshDrawCommands=Empty } DrawList=Empty GraphicsMinimalPipelineStateSet=...}	FMeshPassDrawListContext * {FD...
[FDynamicPassMeshDrawListContext]	{DrawListStorage={MeshDrawCommands=Empty } DrawList=Empty GraphicsMinimalPipelineStateSet={...} ...}	FDynamicPassMeshDrawListCont...
FMeshPassDrawListContext	{...}	FMeshPassDrawListContext
DrawListStorage	{MeshDrawCommands=Empty }	FDynamicMeshDrawCommandSt...
MeshDrawCommands	Empty	TChunkedArray<FMeshDrawCom...
DrawList	Empty	TArray<FVisibleMeshDrawComm...
[原始视图]	{AllocatorInstance={Data=0x00000bf6d9448d68 {MeshDrawCommand=0x00000bf6d9448fc0 {ShaderBindings={ShaderLayouts=...} ...	TArray<FVisibleMeshDrawComm...
GraphicsMinimalPipelineStateSet	{...}	Experimental::TRobinHoodHashS...
NeedsShaderInitialisation	false	bool &
_vfptr	0x00007ffc63a4b270 {UnrealEditor-Renderer.dll!void(* FDynamicPassMeshDrawListContext::`vftable'[3])()} {...}	void * *
MeshBatch	{Elements=Num=1 VertexFactory=0x00000bf684199b00 {Data={PreSkinPositionComponent={VertexBuffer=0x0000000000000000 <...	const FMeshBatch &
MeshBatch.Elements	Num=1	const TArray<FMeshBatchElemen...
MeshDrawCommand	{ShaderBindings={ShaderLayouts={...} Data={InlineStorage=0x0000000000000028 {???, ???, ???, ???, ...} } ...} ...}	FMeshDrawCommand &
NumElements	1	const int
SharedMeshDrawCommand	{ShaderBindings={ShaderLayouts=Num=1 Data={InlineStorage=0x0000005155485c78 {0x0000000000000000 <NULL>, ...} } ...} ...}	FMeshDrawCommand
this	0x00000bf6d9448ce0 {PassDrawRenderState={BlendState=0x00000bf69c1ddf80 {Desc={AlphaToCoverageEnable=...} } ...} ...}	FMeshPassProcessor * {FDepthPa...

在执行完 `AddCommand` 和 `FinalizeCommand` 之后，DepthPass的 `MeshProcessor` 中就有了Cube对应的 `FMeshBatch` 所生成的 `FMeshDrawCommand` 的信息。

名称	值	类型
this	0x00000bf6d9448ce0 {PassDrawRenderState={BlendState=0x00000bf69c1ddf80 {Desc={AlphaToCoverageEnable=...} } ...} ...}	FDepthPassMeshProcessor *
TConcurrentLinearObject<FDepthPassMesh...	{...}	TConcurrentLinearObject<FDept...
FMeshPassProcessor	{MeshPassType=DepthPass (0 '\0') Scene=0x00000bf6d4edb000 {World=0x00000bf6d4fb7800 (Name="RenderAnalysisMap"...	FMeshPassProcessor
IPSOCollector	{...}	IPSOCollector
MeshPassType	DepthPass (0 '\0')	EMeshPass::Type
Scene	0x00000bf6d4edb000 {World=0x00000bf6d4fb7800 (Name="RenderAnalysisMap") FXSystem=0x00000bf6d3c4fed0 {...} ...}	const FScene *
FeatureLevel	SM6 (4)	ERHIFeatureLevel::Type
ViewIfDynamicMeshCommand	0x00000bf6de0dc000 {ViewRect={Min={X=0 Y=0 XY=0x00000bf6de0de578 {0, 0} } Max={X=1288 Y=724 XY=0x00000bf6de0d...	const FSceneView * {FViewInfo}
DrawListContext	0x00000051554861a0 {DrawListStorage={MeshDrawCommands=NumElements=1, NumChunks=1, NumElementsPerChunk ...	FMeshPassDrawListContext * {FD...
[FDynamicPassMeshDrawListContext]	{DrawListStorage={MeshDrawCommands=NumElements=1, NumChunks=1, NumElementsPerChunk (48) } DrawList=Num...	FDynamicPassMeshDrawListCont...
FMeshPassDrawListContext	{...}	FMeshPassDrawListContext
DrawListStorage	{MeshDrawCommands=NumElements=1, NumChunks=1, NumElementsPerChunk (48) }	FDynamicMeshDrawCommandSt...
MeshDrawCommands	NumElements=1, NumChunks=1, NumElementsPerChunk (48)	TChunkedArray<FMeshDrawCom...
DrawList	Num=1	TArray<FVisibleMeshDrawComm...
[0]	{MeshDrawCommand=0x00000bf6b3824000 {ShaderBindings={ShaderLayouts=Num=1 Data={InlineStorage=0x00000bf6b3...	FVisibleMeshDrawCommand
MeshDrawCommand	0x00000bf6b3824000 {ShaderBindings={ShaderLayouts=Num=1 Data={InlineStorage=0x00000bf6b3824020 {0x0000000000000...	const FMeshDrawCommand *
ShaderBindings	{ShaderLayouts=Num=1 Data={InlineStorage=0x00000bf6b3824020 {0x0000000000000000 <NULL>, 0x0000000000000000 <...	FMeshDrawShaderBindings
VertexStreams	Num=3	TArray<FVertexInputStream,TSize...
IndexBuffer	0x00000bf6d146ba00 {LockedData={ResourceLocation={Owner=0x0000000000000000 <NULL> UnderlyingResource=...} ...} }	FRHIBuffer * {UnrealEditor-D3D12...
CachedPipelineId	{PackedId=3221225472 SetElementIndex=0 bComesFromLocalPipelineStateSet=1 ...}	FGraphicsMinimalPipelineStateId
FirstIndex	0	unsigned int
NumPrimitives	48	unsigned int
NumInstances	1	unsigned int
VertexParams	{BaseVertexIndex=0 NumVertices=54 }	FMeshDrawCommand::<unname...
IndirectArgs	{Buffer=0x0000003600000000 {Size=??? Stride=??? Usage=??? } Offset=1675720720 }	FMeshDrawCommand::<unname...
PrimitiveIdStreamIndex	2 '\x2'	char
StencilRef	0 '\0'	unsigned char
PrimitiveType	PT_TriangleList (0)	EPrimitiveType
DebugData	{PrimitiveSceneProxyIfNotUsingStateBuckets=0x00000bf6a1170c00 {RenderData=0x00000bf68419a000 {LODResources=...}	FMeshDrawCommandDebugData
SortKey	{PackedData=57657 BasePass={VertexShaderHash=57657 PixelShaderHash=0 Masked=0 } Translucent={MeshIdInPrimitive=...	FMeshDrawCommandSortKey
PrimitiveIdInfo	{DrawPrimitiveId=0 ScenePrimitiveId=0 InstanceSceneDataOffset=0 ...}	FMeshDrawCommandPrimitiveId...
PrimitiveIdBufferOffset	-1	int
StateBucketId	-1	int
RunArray	0x0000000000000000 {???}	const unsigned int *

Footnote

FMeshBatchElement

FMeshBatchElement里面储存了单个网格所需的数据，如IndexBuffer，shaderParameters等

FMeshBatch

FMeshBatch包含了一个pass的所需要的全部渲染数据，它会维护一个FMeshBatchElement列表，FMeshBatchElement包含了单个网格绘制所需的数据，包括UniformBuffer、IndexBuffer等等。事实上，最后一个FMeshBatchElement就对应了一次DrawCall
FMeshBatch解耦了Pass和FPrimitiveSceneProxy，包含了绘制pass所需信息
他拥有一组FmeshBatchElement（但绝大多数情况下只用一个，除了FInstancedStaticMeshSceneProxy和FHierarchicalStaticMeshSceneProxy中的接口会对数组作填充，其余情况都只用到一个FMeshBatchElement），他们共享相同材质的vertexFactory

FMeshElementCollector

FMeshElementCollector 由 FSceneRenderer 创建且一一对应。
收集器收集完对应view的可见图元列表后，通常拥有一组需要渲染的FMeshBatch列表，以及它们的管理数据和状态，为后续的流程收集和准备足够的准备。
此外，FMeshElementCollector在收集完网格数据后，还可以指定需要等待处理的任务列表，以实现多线程并行处理的同步。

GetDynamicMeshElements() & GetDynamicElementsSection()

```
void FSceneRenderer::GatherDynamicMeshElements(){
    PrimitiveSceneInfo->Proxy->GetDynamicMeshElements();
}
```

是给每个图元对象向渲染器（收集器）添加可见图元元素的机会，由具体的子类实现，如FSkeletalMeshSceneProxy
FSkeletalMeshSceneProxy会根据不同的LOD索引，给每个Section网格添加一个FMeshBatch。

ParallelMeshDrawCommandPasses

```
void FParallelMeshDrawCommandPass::DispatchPassSetup()
{
    ... 先收集 TaskContext 信息 ...
    FMeshDrawCommandPassSetupTask::AnyThreadTask() 使用 TaskContext 信息生成绘制指令、写入数据
    FMeshDrawCommandInitResourcesTask::AnyThreadTask() 使用 TaskContext 信息初始化绘制资源
}
```

FMeshDrawCommandPassSetupTaskContext

收集FMeshDrawCommandPassSetupTask需要的信息

FMeshDrawCommandPassSetupTask

在FMeshDrawCommandPassSetupTask中进行绘制指令生成与相关数据的写入

FMeshPassProcessor & AddMeshBatch() & TryAddMeshBatch() & BuildMeshDrawCommands() & FMeshPassDrawListContext

每个Pass都对应了一个FMeshPassProcessor，每个FMeshPassProcessor保存了该Pass需要绘制的所有FMeshDrawCommand，以便渲染器在合适的时间触发并渲染。
不同Pass的通过调用AddMeshBatch()方法处理FMeshBatch中的几何信息，主要的处理在TryAddMeshBatch()中，该方法中进行了shader绑定，渲染转台处理等，最后根据不同的选项和质量选择不同的Process使用BuildMeshDrawCommands()将FMeshBatch转为FMeshDrawCommand
生成的FMeshDrawCommand被保存在FMeshPassDrawListContext中

GenerateDynamicMeshDrawCommands()

转换指定EMeshPass中的每个FMeshBatch到一组FMeshDrawCommand。
FMeshDrawCommandPassSetupTask要用到。

```
void GenerateDynamicMeshDrawCommands(){
    PassMeshProcessor->AddMeshBatch();
}
```

FMeshDrawCommand

内有资源绑定信息如着色器绑定(ShaderBindings)、顶点流(VertexStreams)、索引缓冲(IndexBuffer)、PSO管线ID(CachedPipelineId)、绘制参数(FirstIndex、NumPrimitive、NumInstances)等。

> FMeshDrawCommand（网格绘制指令），记录了绘制单个Mesh所需的所有资源和数据，且不应该有多余的数据，如果需要在InitView传递数据，可用FVisibleMeshDrawCommand。

FParallelMeshDrawCommandPass & DispatchPassSetup() & DispatchDraw()

Encapsulates two parallel tasks - mesh command setup task and drawing task
DispatchPassSetup() 对应 mesh command setup task
DispatchDraw() 对应 drawing task
同时保存着该pass的meshdrawcommand

RHICommand

RHI全称Rendering Hardware Interface（渲染硬件接口），是不同图形API的抽象层，而RHICommandList便是负责收录与图形API无关的中间层绘制指令和数据。

RHICommandList

RHICommandList收录了一系列中间绘制指令之后，会在RHI线程一一转换到对应目标图形API的接口

RDG

RDG全称是Rendering Dependency Graph，意为渲染依赖性图表，是UE4.22开始引进的全新的渲染子系统，基于有向无环图(Directed Acyclic Graph，DAG)的调度系统，用于执行渲染管线的整帧优化。UE中使用RDG代替原本直接调用RHI命令的方式，由RDG调整资源的生命周期，裁剪Pass，处理Pass的资源转换和屏障，处理异步计算Pass的依赖和引用关系，查找并建立分叉和合并Pass节点，合并所有具体相同渲染目标的光栅化Pass等。

FRDGPass

RDGPass和渲染Pass并非一一对应关系，有可能多个RDGPass合并成一个渲染Pass。

Reference

0. 虚幻引擎网格体绘制管道 | 虚幻引擎5.2文档
1. 剖析虚幻渲染体系（03）- 渲染机制 - 0向往0 - 博客园
2. UE5【理论】1.网格绘制管线MeshDrawPipeline
3. 虚幻4渲染编程(Shader篇)【第十二卷：MeshDrawPipline】
4.