Analysis of Unreal Rendering System (06) - UE5 Special Part 1 (Features and Nanite)

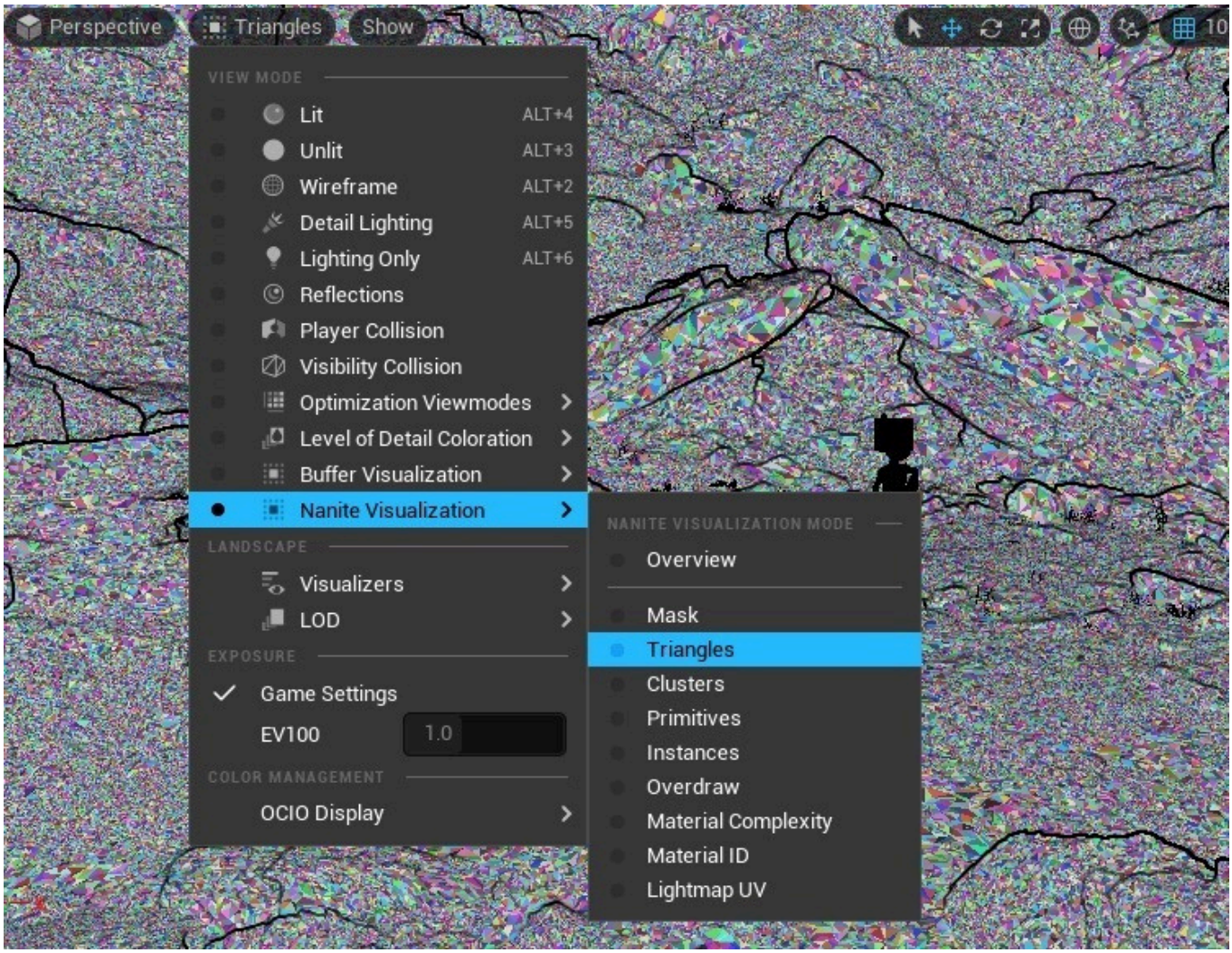
Table of contents

6.1 Overview

As early as May 2020, Unreal officially released a video**Lumen in the Land of Nanite**that showcased the rendering features of Unreal Engine 5. The video showcased Nanite, a virtual micropolygon geometry-based Lumen technology, and real-time global illumination, bringing a film-level audio-visual experience to real-time games.

At that time, Unreal officially promised to release a preview version of UE5 in the first half of 2021. Sure enough, it kept its promise and successfully released a preview version of**UE5 Early Access (EA)**in late May 2021. So, we can study UE5's editor, tool chain, new rendering features, and the corresponding UE5 EA version source code and the accompanying resource project AncientGame.

(The following few useless pages about creating a project with nanite have been removed.)



The Lit mode of the level editor has added a Ninite visualization group. The background noise is not a bug, it shows the triangle mode of Nanite.

6.2.2 New rendering features

This section will explain the new rendering features of UE5.

6.2.2.1 Nanite Virtual Micropolygons

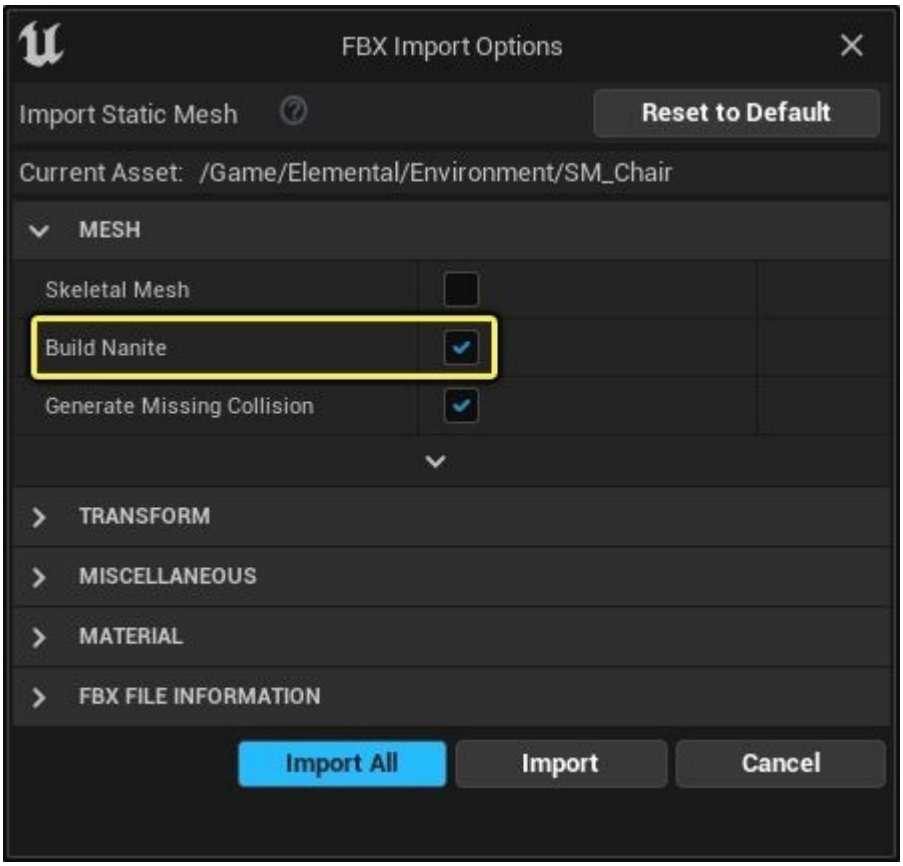
Nanite means nanorobot. UE5 uses it to name the new generation of mesh processing shading technology. The intention is obvious, which is to replace and upgrade the traditional culling and rasterization shading technology based on mesh LOD granularity, and use extremely small granularity to process meshes and triangles.

UE5's Nanite stands for**Nanite Virtualized Geometry**(Nanite virtual micro-geometry, Nanite virtual micro-polygons), which supports automated processing of high-precision mesh models, high-detail surfaces of pixel-level triangles, and massive objects. It will only process the required and only required data at the appropriate level to prevent the loss of surface details or the processing of too much data. Nanite performs a lot of pre-processing on meshes, textures, animations and other data before rendering, saves them in highly compressed and fine-grained binary streams, and automatically processes their LODs.

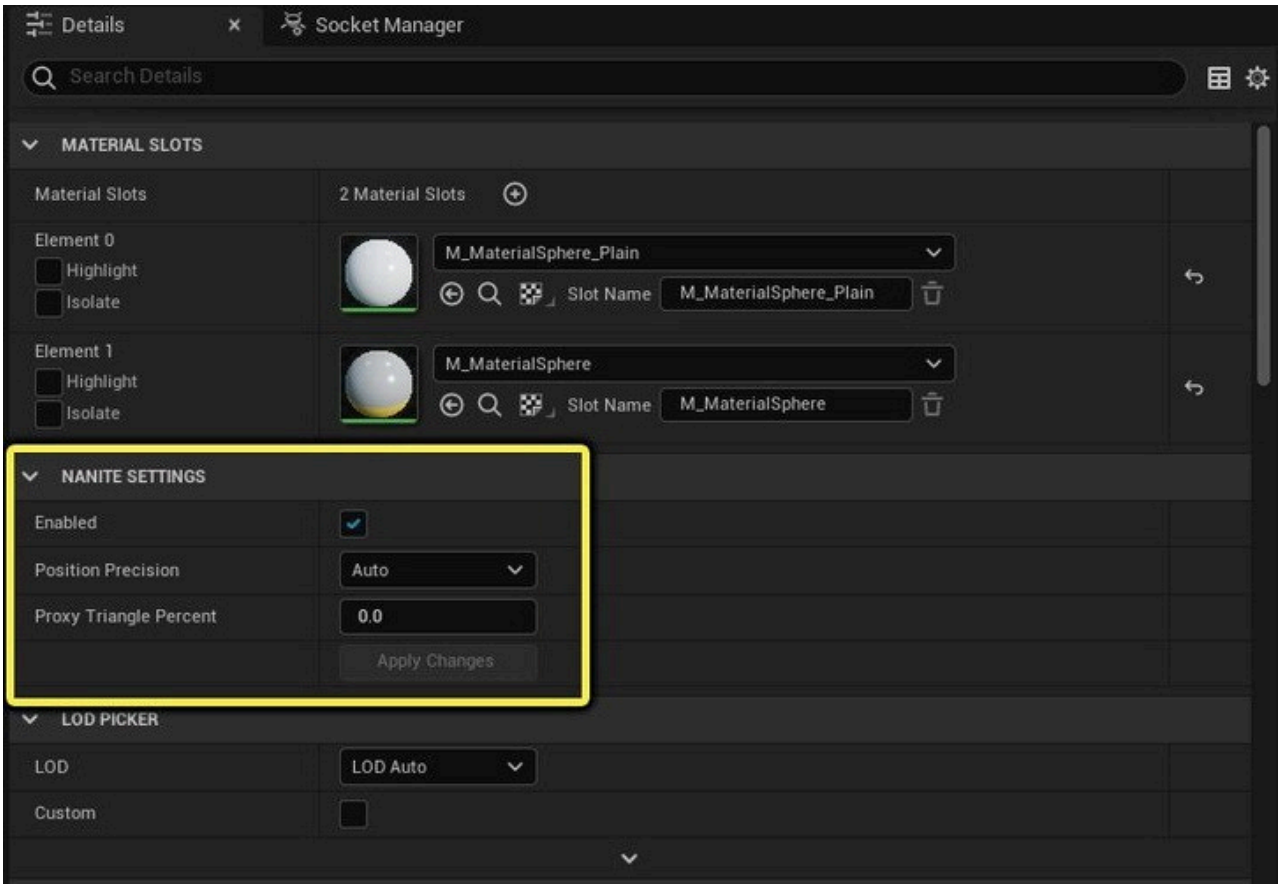
Overview of the Nanite micropolygon technology of the AncientGame sample project. Top left: AncientGame's temple; top right: the corresponding micropolygon visualization of the top left; bottom left and bottom right are the project's boss and mountain details respectively.

There are three ways to enable Nanite technology for UE5 mesh:

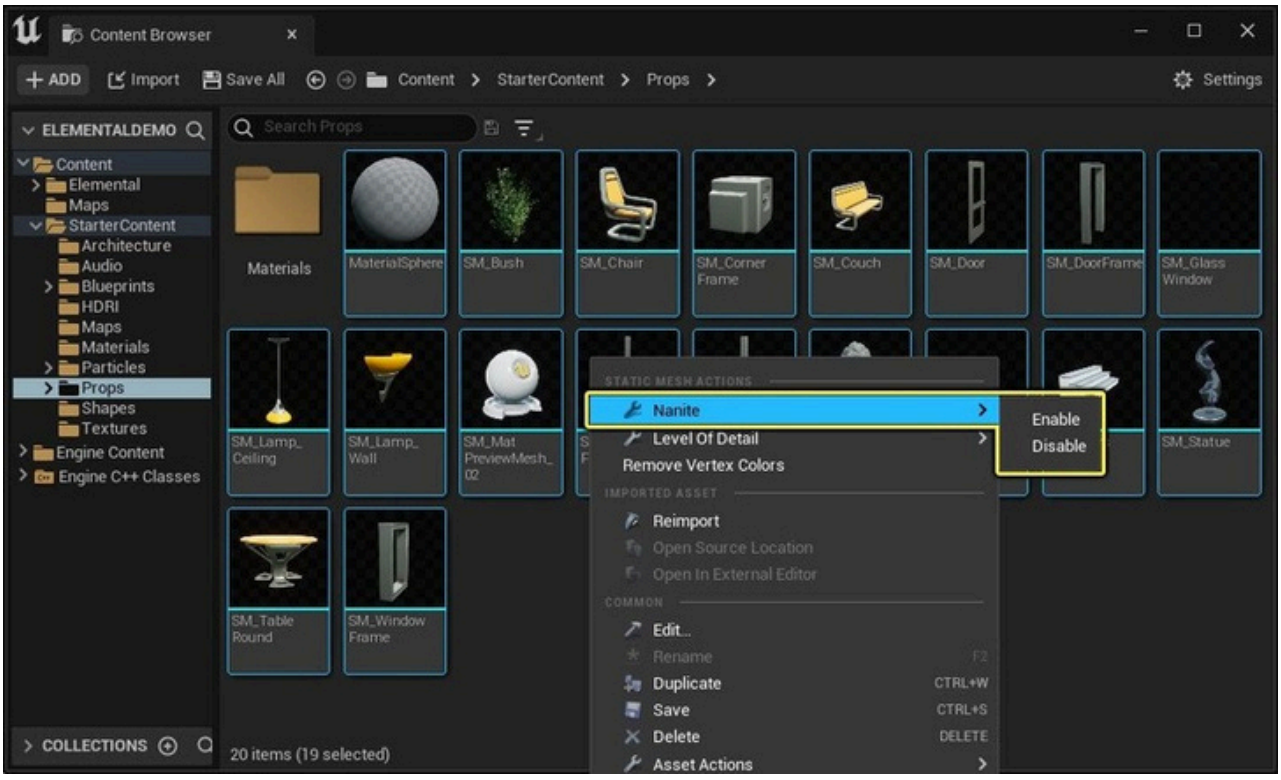
- This can be turned on in the settings interface when importing a static mesh.



- This can be set in the Mesh Properties panel of the Mesh Editor.



- You can open them in batches through the right-click menu of the resource browser.



After enabling Nanite technology, you can get many benefits:

- Supports orders of magnitude more geometric complexity, higher numbers of triangles and objects than UE4, supporting real-time processing and rendering.
- Real-time frame performance is no longer limited by polygon count, draw calls, and mesh memory usage.

- (Possibly) support direct import of high-precision models made from DCC software such as ZBrush.
- Use high-resolution meshes directly instead of using pre-baked normal maps.
- LOD is automatically processed, and there is no need to manually set up and process mesh LOD.

- There is no or only a small amount of quality loss during LOD transition.

Nanite mesh is similar to traditional static mesh, it is still a triangle mesh in essence, but its core difference is a large amount of details and high data compression. Most importantly, Nanite uses a brand new system to render data format in an extremely efficient way.

Traditional static meshes require a flag to enable Nanite technology. Nanite meshes can support multiple sets of UVs and vertex colors. Materials are assigned to parts of the mesh, so these materials can use different shading models and dynamic effects (which can be performed in the shader). Material assignments can be swapped dynamically, just like any other static mesh, and Nanite requires no additional processing to bake materials.

Since Nanite has better rendering performance and takes up less memory and disk space, it is best to enable the Nanite properties of static meshes as much as possible. In order to better utilize Nanite technology, static meshes should meet the following requirements:

- Contains a large number of triangles or the triangles occupy a very small size on the screen.
- Having many instances in the same scene.
- Can be used as an occluder to block other Nanite geometries.

At present, Nanite supports the following component types:

- Static Mesh
- Instanced Static Mesh
- Hierarchy Instanced Static Mesh
- Geometry Collection

Nanite does not support animation types including but not limited to:

- Skeletal animation
- Morph Targets (Blend Shape) World
- Position Offset in materials Spline
- meshes

In addition, Nanite does not support the following features:

- Custom depth or stencil
- Vertex painting on instances

It should be noted that the vertex tangents of Nanite meshes are not stored in the mesh data like traditional static meshes (the official documentation explains that this is to reduce the data size), so

the tangents are calculated dynamically in the pixel shader. Nanite may cause discontinuities at the edges due to the difference in the use of tangent space and traditional methods.

Nanite also does not support materials with the following configurations:

- Blend Mode other than Opaque.
    - Masked and semi-transparent ones are not supported.
- Delayed decals.
- Wireframe mode.
- Pixel depth offset.
- World position offset.
- Customize per-instance data.
- Double-sided material.

Nanite will not render properly (may disappear) materials that use the following features:

- Vertex Interpolator node.
- Custom UVs.

Nanite does not support the following rendering features:

- The following view-dependent object filters are used:
    - Minimum screen radius
    - Distance Clipping
    - Filtering with FPrimitiveSceneProxy::IsShown()
    - Scene Capture with the following features:
        - Hiding Components
        - Hiding Actors
        - Display only components
        - Actors Shown Only
- Forward Rendering
- Stereo rendering in VR mode
- split-screen
- MSAA
- Lighting channel
- Ray tracing of a fully detailed Nanite mesh
- Partial visualization mode
-

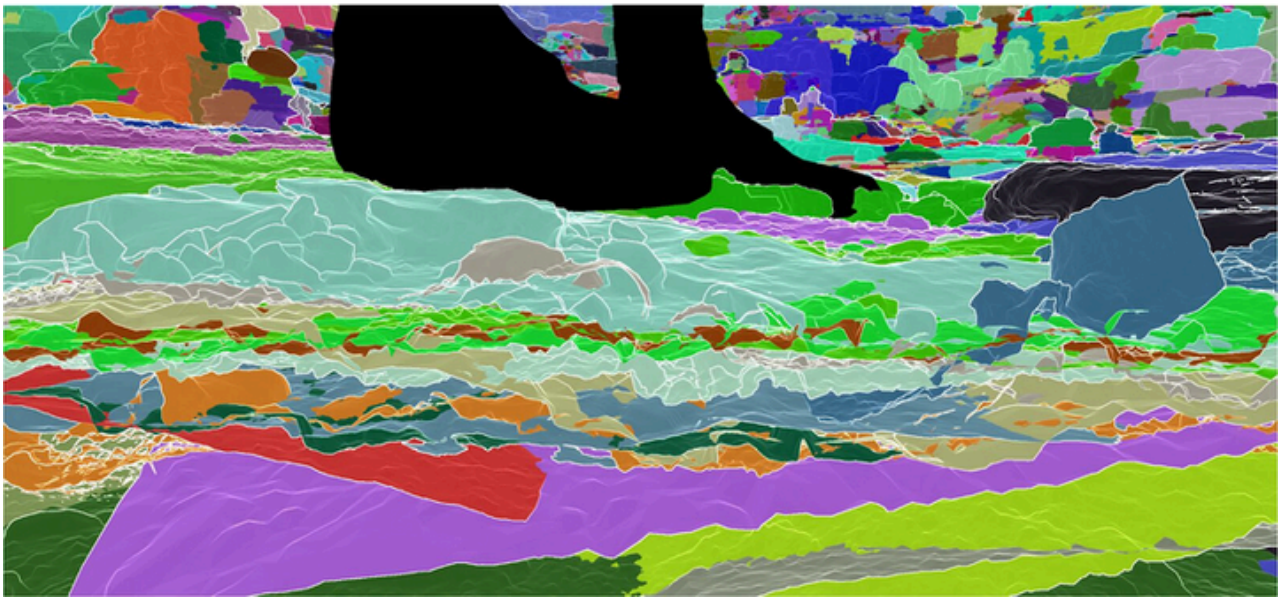
Nanite supports the following GPUs on PlayStation 5, Xbox Series S|X, PC, and other platforms with the latest drivers:

- NVIDIA: Maxwell or newer graphics cards

- AMD: GCN or newer graphics cards

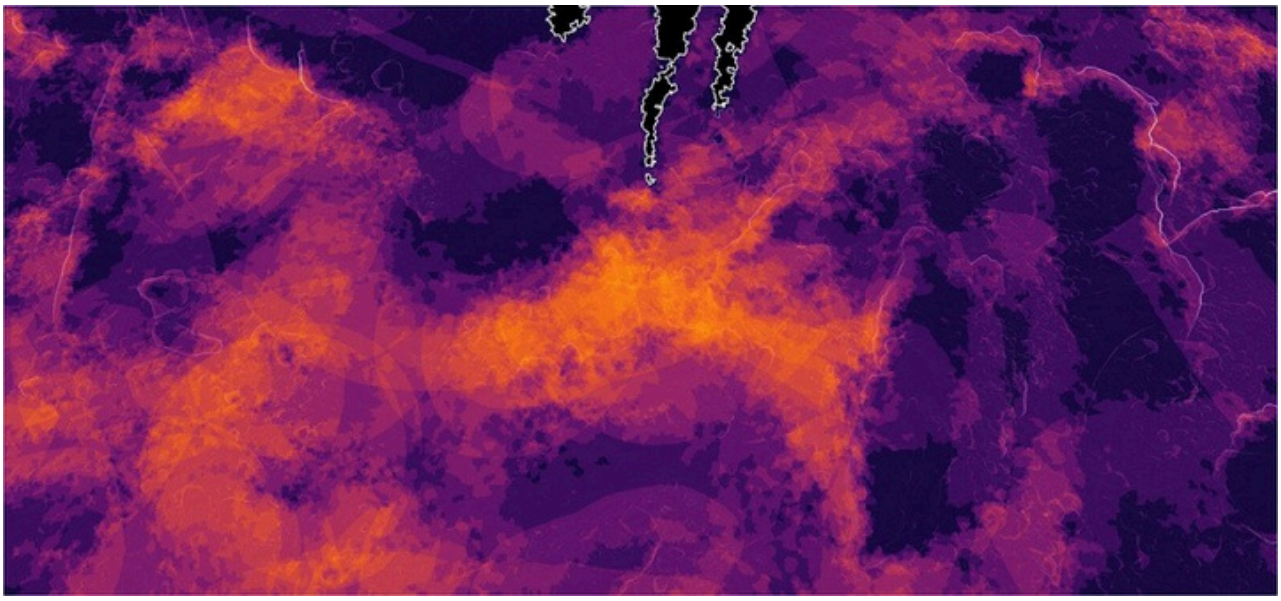
In order to monitor performance, Nanite needs to pay attention to the following points:

- **Aggregate Geometry**:Aggregate Geometry combines many tiny, unconnected things into a volume in the distance, such as hair, leaves, and grass. It affects LOD and occlusion culling techniques.
- **Closely Stacked Surfaces**:Nanite will merge objects that are close to the top-level surface of the view and draw all stacked objects without considering the occlusion and hiding relationships between them.



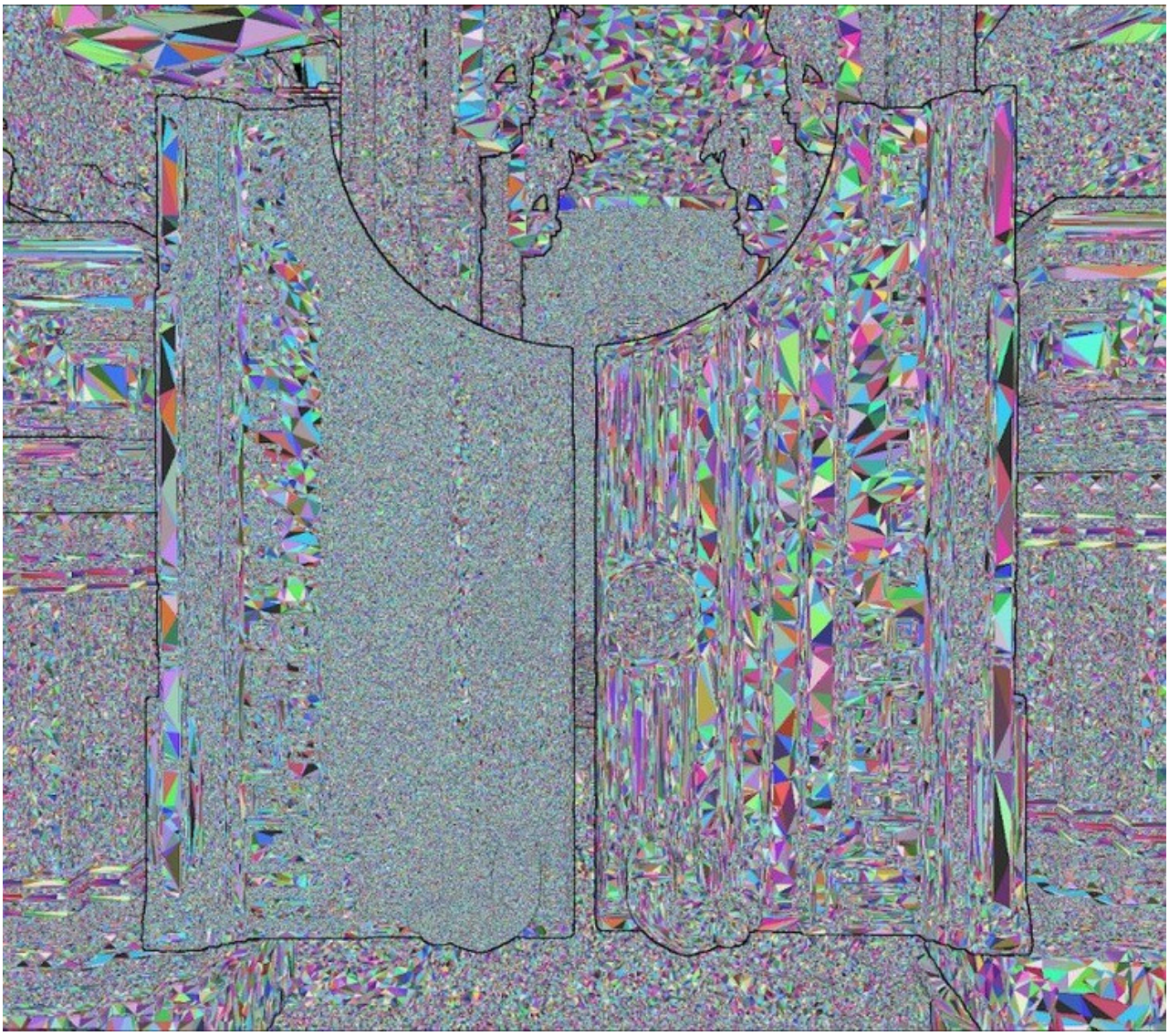
The above picture is the normal screen of AncientGame, and the picture below is the corresponding instance visualization using Closely Stacked Surfaces. The black part is because Nanite does not support dynamic characters.

In most cases, Closely Stacked Surfaces will reduce draw calls, but in some cases it may have the opposite effect. Moving the camera in Overdraw visualization mode can show how these stacked surfaces are rendered:



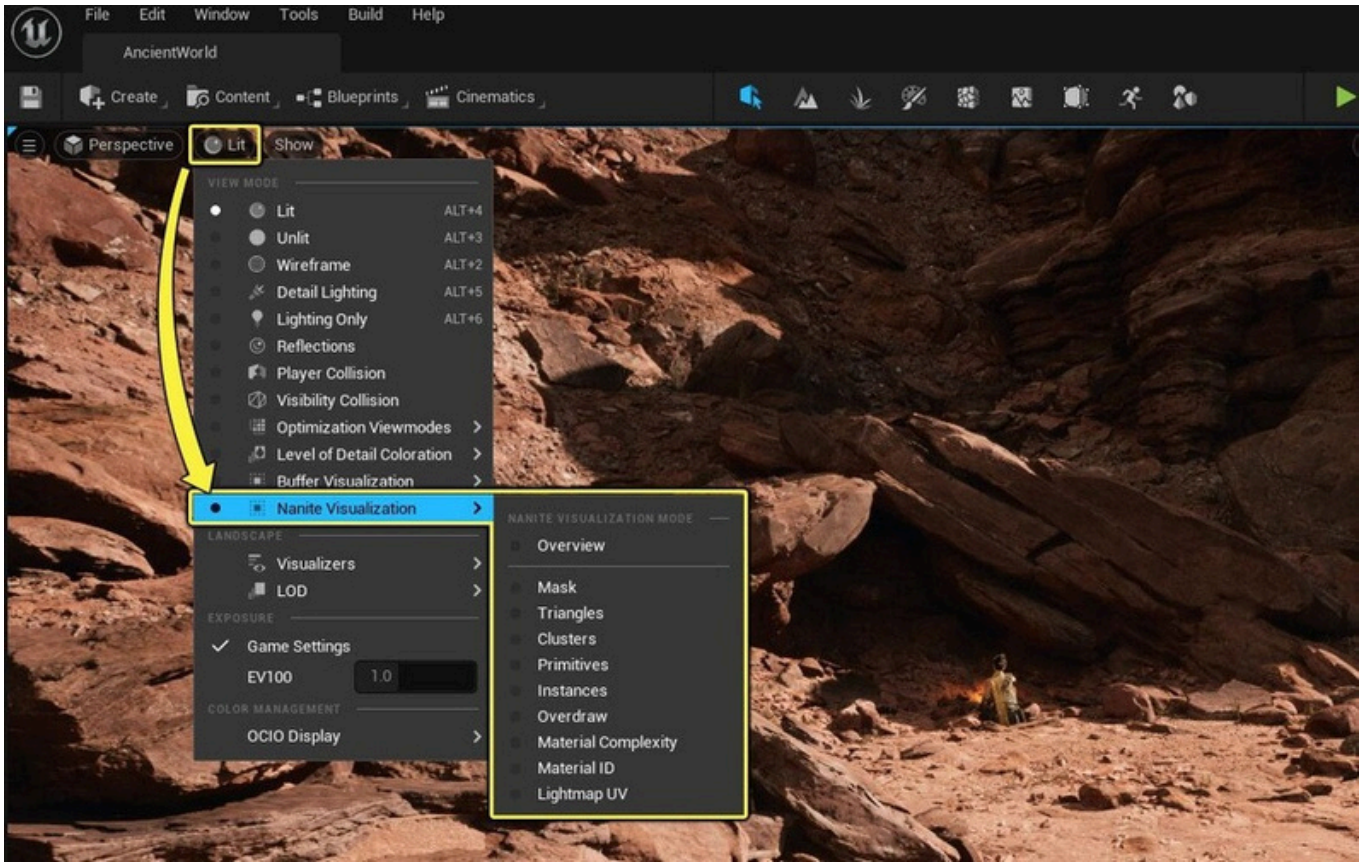
- **Faceted and Hard-edge Normals**:Ideally, your mesh should have fewer vertices than triangles. If the ratio of vertices to triangles is 2:1 or higher, there may be problems, especially if the triangle count is high. A ratio of 3:1 means that the mesh is fully faceted, where each triangle has its own three vertices, none of which are shared with another triangle, usually because they have different normals due to lack of smoothing. The following two images show the similarities and differences between faceted and smoothed normals:

The left side of the above

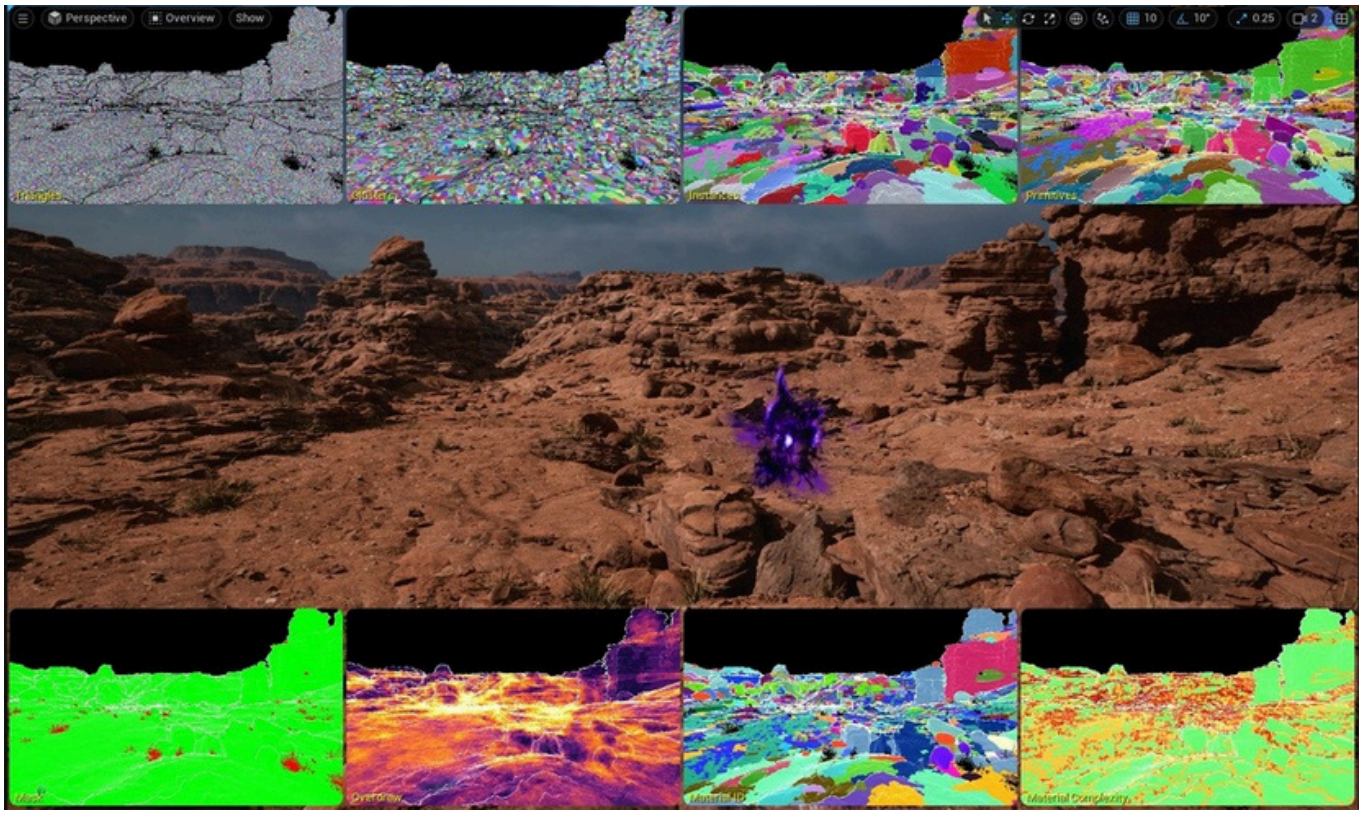


The left image above shows a Nanite triangle visualization using facet normals, and the right image above shows a Nanite triangle visualization using smoothing group normals. It can be seen that normals using smoothing groups will use fewer triangles to draw.

In addition to the visualization modes mentioned above, UE5 also provides a variety of other visualization methods. You can view all data through the Overview mode:

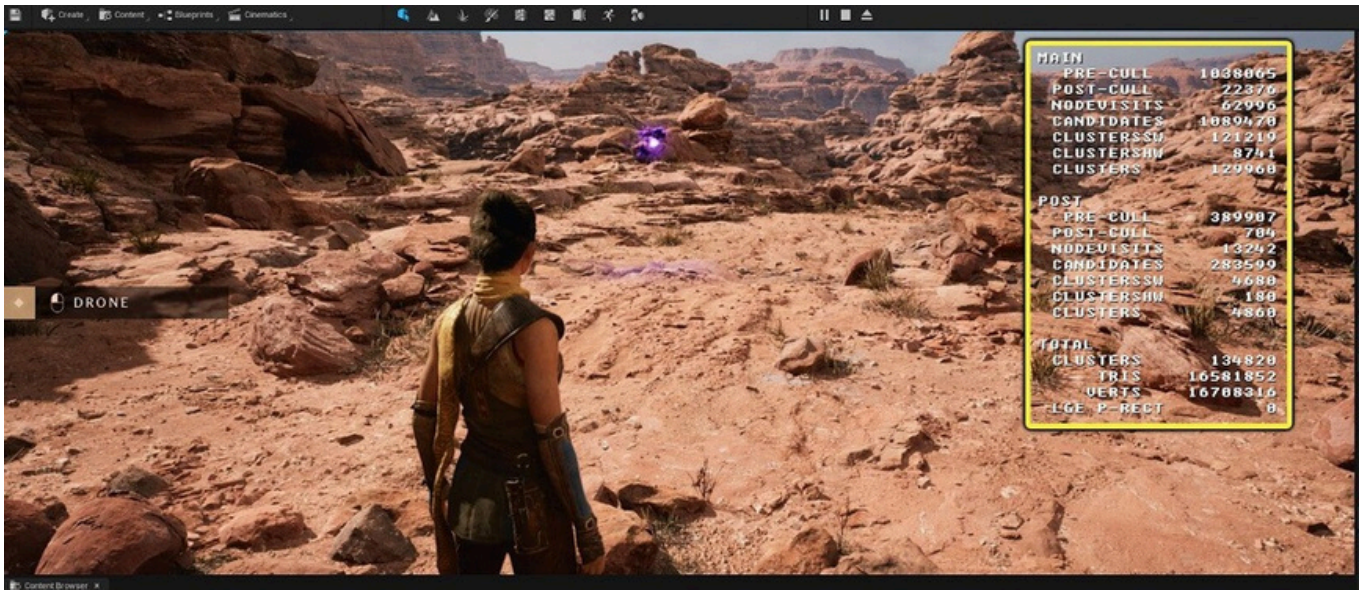


UE5 provides a variety of visualization modes for Nanite, each displaying different data.



Opening Nanite's Overview will display all visualization mode thumbnails.

Using console variables Nanitestats, you can view Nanite statistics of the current screen in real time:

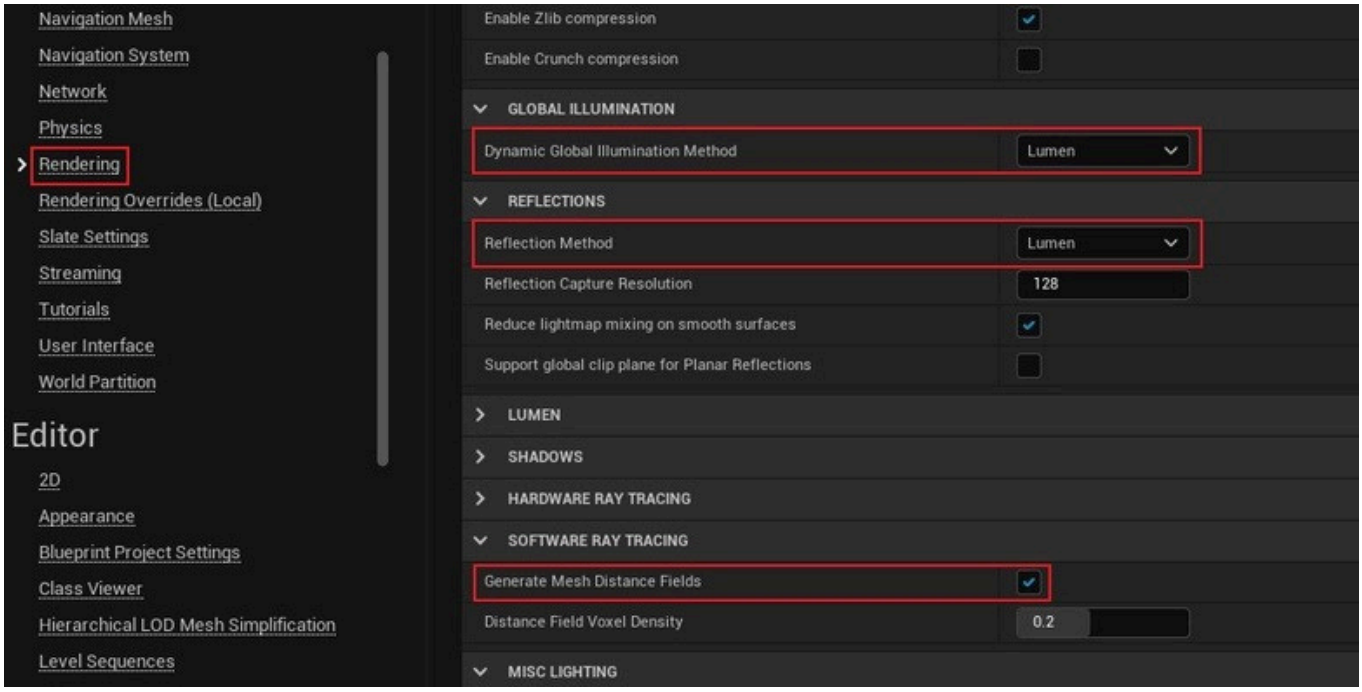


The right side shows the clipping data and geometry data of Nanite.

6.2.2.2 Lumen global dynamic lighting

Lumenis UE5's fully dynamic global illumination and reflection system, which is the default global illumination and reflection method for UE5. Lumen can present diffuse reflections with infinite bounces and indirect specular reflections in large-scale, detailed environments ranging from millimeters to kilometers.

To enable Lumen, you need to enable the following options in the project settings (all enabled by default):



After turning on Lumen, Lumen GI will replace SSGI and DFAO, Lumen reflections will replace SSR, and static lighting will be disabled, and all light maps will be hidden.

The rendering features supported by Lumen are as follows:

- **Lumen Global Illumination**

Lumen GI solves the indirect lighting of scene objects. For example, pixels of direct light will affect nearby pixels, a phenomenon also known as color bleeding. At the same time, since the grid will block and absorb some indirect light, Lumen can also correctly handle the shadow occlusion of indirect light.

Lumen global illumination can dynamically handle the lighting and shadowing effects of indirect light in real time.

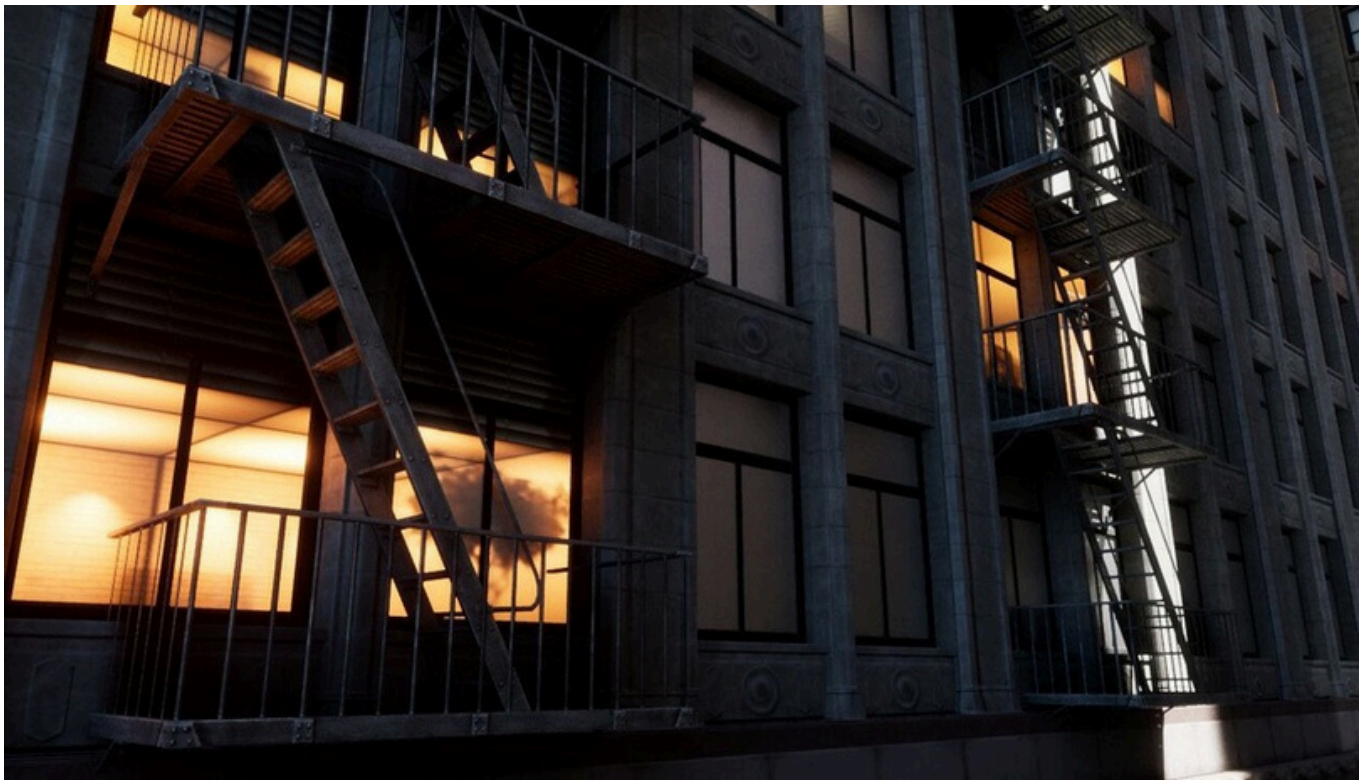
Lumen achieves full-resolution normal detail while calculating indirect lighting at a lower resolution for real-time rendering purposes.

- **Lumen Sky Light**

Lumen solves sky lighting in the Final Gather stage, making the sky light indoors and outdoors have a clear difference, and it is darker indoors. In addition, Lumen's sky light also supports low-quality transparent lighting and volumetric fog GI effects.

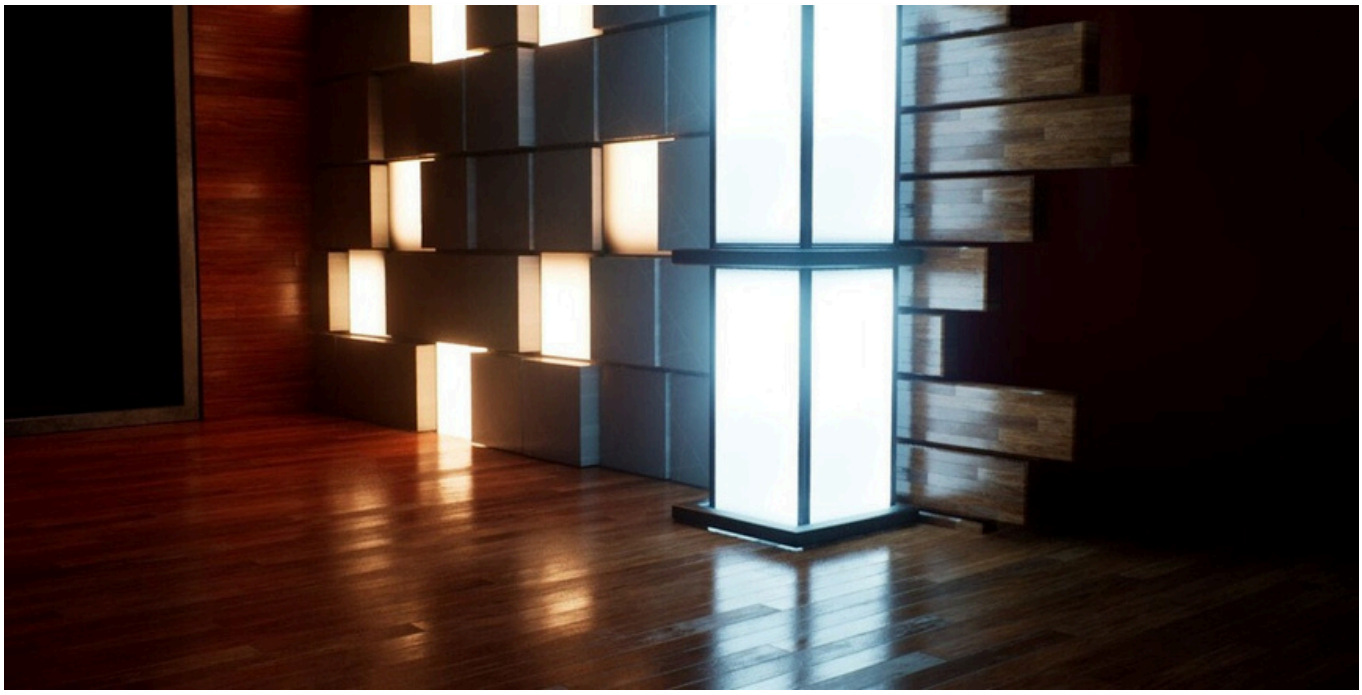
- **Lumen**

The luminous material completes the light propagation through Lumen's Final Gather without any additional consumption. However, there are also restrictions on the size and brightness of the radiant area of the luminous material, otherwise it will cause noise artifacts.



- **Lumen Reflection**

Lumen solves indirect specular effects for materials of all ranges of roughness.



Additionally, Lumen also supports reflections in clearcoat materials.

In the EA version, Lumen's support for light source features is as follows:

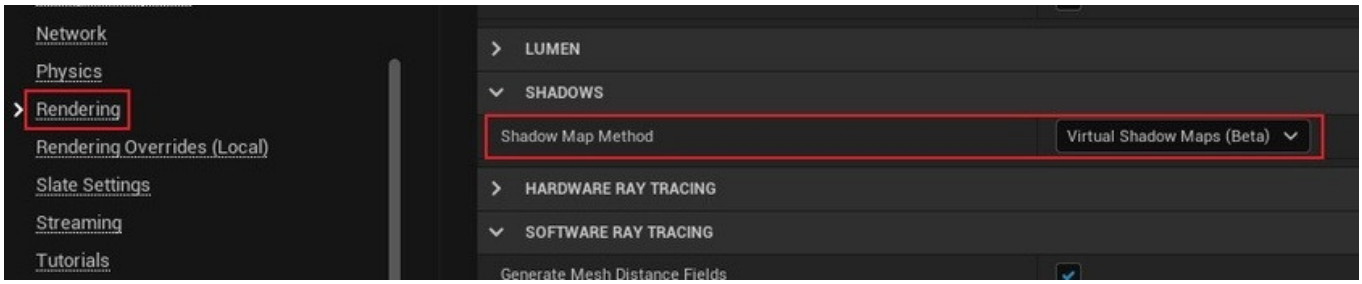
- Supports all light types, including directional light, point light, spotlight, rectangular light, and sky light.
- Light functions for directional lights are only supported.
- [Not] Supports static properties of light sources, because turning on Lumen will disable static lights and light maps.

In the project settings and post-processing volume, you can set many parameters of Lumen, such as soft ray tracing mode, detail tracking mode, global tracking mode, hardware tracking mode, as well as GI and reflection.

6.2.2.3 Virtual Shadow Map

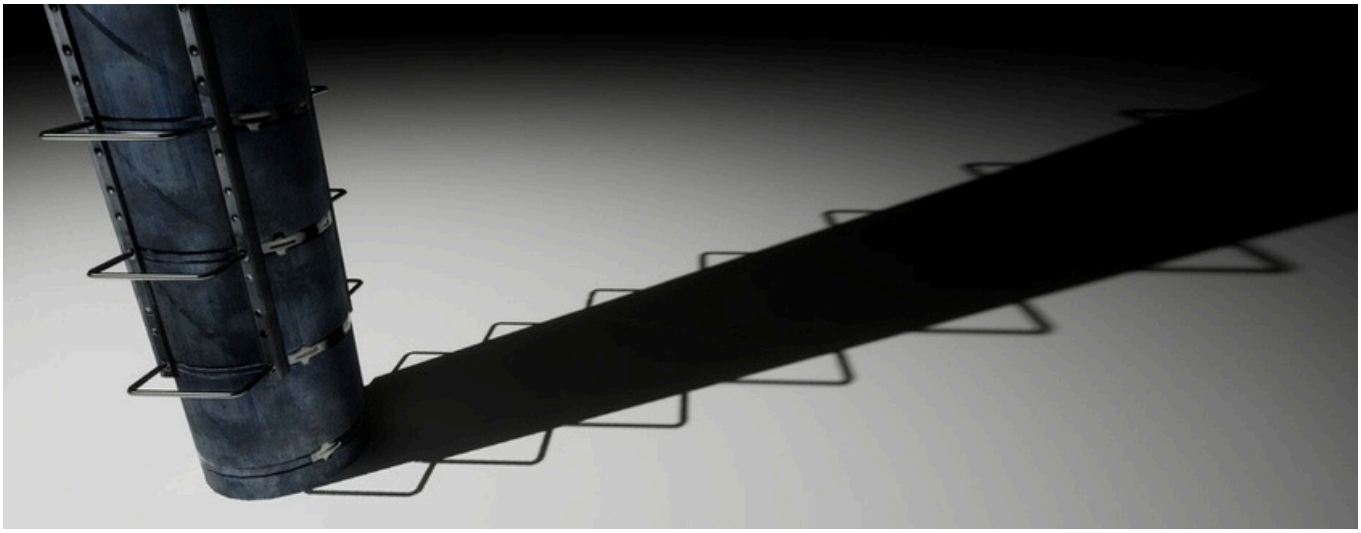
Virtual Shadow Map (VSM)is a new shadow casting method for providing consistent, highresolution shadows, comparable to movie-quality assets and dynamic lighting for large open worlds.

Enable VSM in the Shadow Map Method in Project Settings (enabled by default):



When VSM is turned on, traditional shadow technologies will be replaced, including fixed precalculated shadows, distance field shadows, preview shadows, per-object shadows, cascade shadows, mobile dynamic shadows, etc.

Once VSM is turned on,**Shadow Map Ray Tracing (SMRT)**can use it to achieve many more accurate and clear shadows and related features, including penumbra, soft shadows, contact hard shadows, etc.



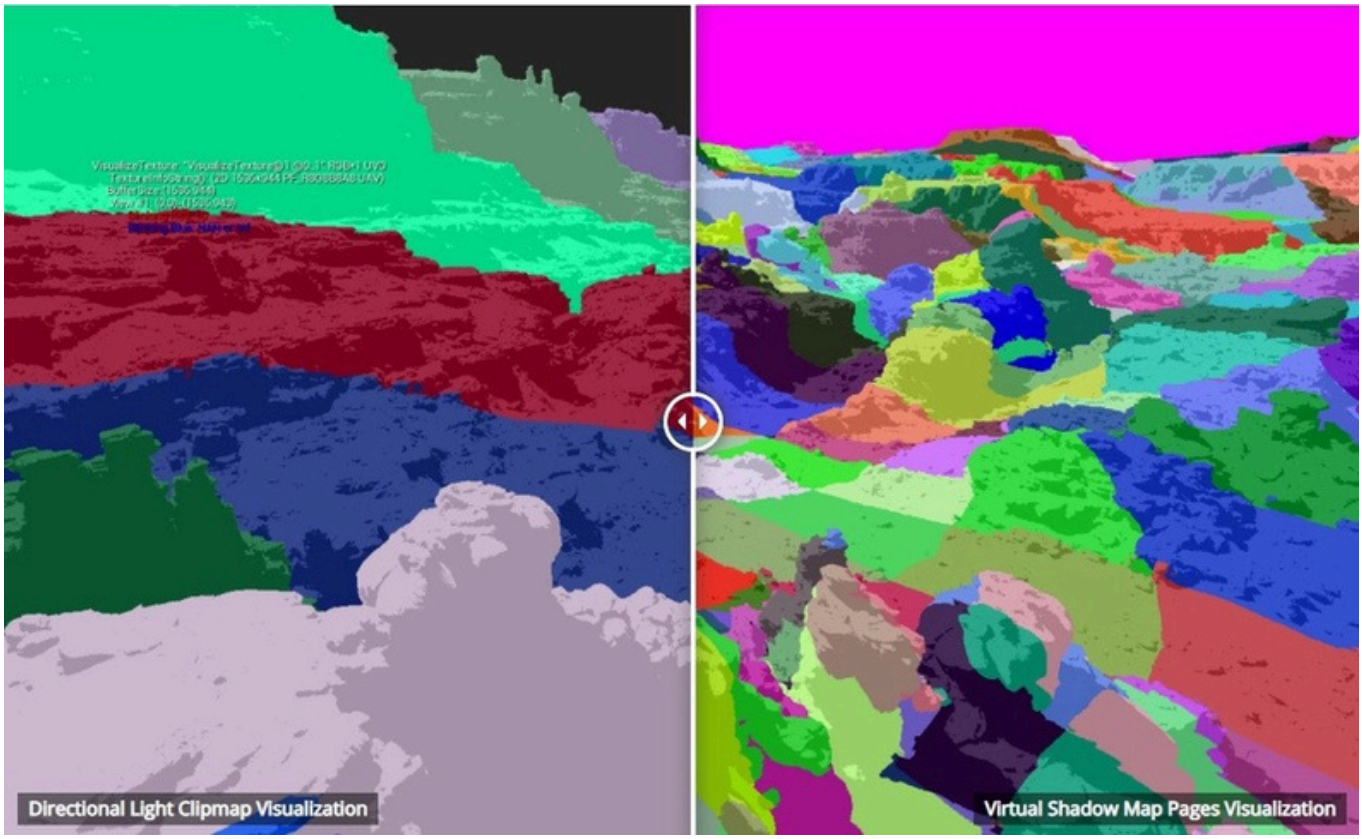
Using SMRT technology, point light sources achieve the characteristics of soft shadows and contact hard shadows.

Left: PCF blurs and removes important surface details; Right: SMRT provides more believable soft shadows and contact hard shadows.

We all know that in traditional rendering, CSM is used to optimize the shadow of directional light. Similarly, UE5 uses**clipmap**technology for directional light.

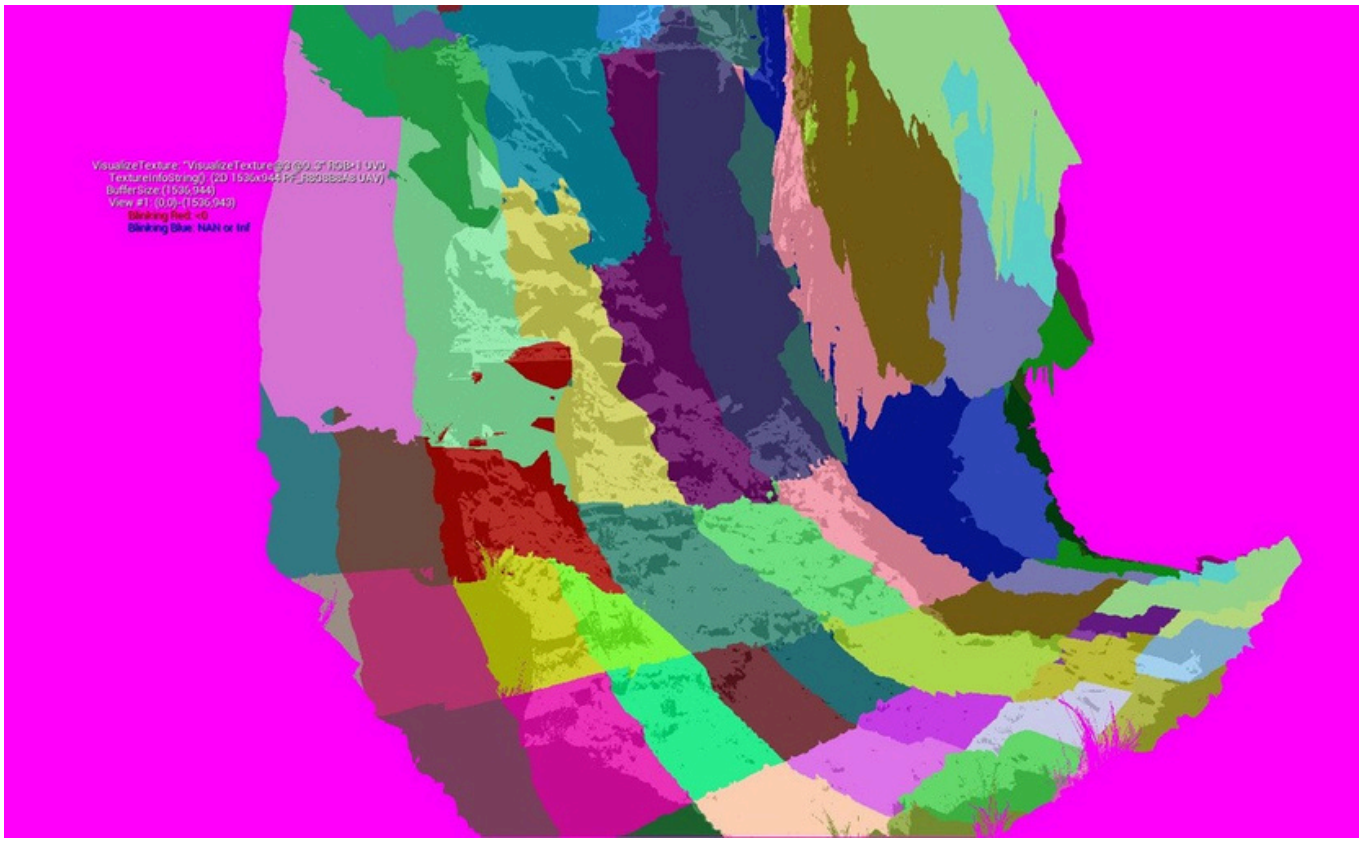
A single virtual shadow map does not provide enough resolution to cover a larger area. Directional lights use a clipmap structure that extends around the camera, with each clipmap level having its

own 16K VSM. Each clipmap level has the same resolution, but covers twice the radius of the previous level.



Left: Clip graph visualization; Right: VSM page visualization.

The spotlight uses a single 16k VSM with mip chain to handle the LOD of the shadow instead of clipmap; the point light uses a cube map, with 16k VSM per face, for a total of 6.



Top: Spotlight effect; Bottom: Corresponding single VSM visualization.

In order to optimize shadow rendering performance, UE5 uses technologies such as caching and coarse pages.

6.2.2.4 Temporal Super-Resolution

Temporal Super Resolution (TSR)is a new generation of temporal anti-aliasing algorithm, which is used to replace the traditional TAA. It supports the following features:

- Utilizing a low-resolution 1080p input, the output is close to native 4K rendering quality, allowing for higher frame rates and better rendering fidelity. Less ghosting in high-frequency
- backgrounds.
- Reduce flickering on highly complex geometry.
- Supports running on Shader Model 5 compatible hardware: D3D11, D3D12, Vulkan, PS5, XSX, Metal is not yet supported.
- The shaders are optimized specifically for the GPU architecture of PS5 and XSX.

Temporal super-resolution can be turned on or off in the project configuration. By default, UE5 has this technology turned on.

Left: 4K native resolution rendering, the frame rate is 20.57; Right: Using temporal super-resolution technology to output 1080p 4K quality, the frame rate is increased to 44.22.

6.2.2.5 Mobile rendering

UE5 has improved some rendering modules for mobile terminals, including: The mobile renderer uses RDG (Rendering Dependency Graph). Support for

- Distance Field Ambient Occlusion and Global Distance Fields.
- DirectX Shader Compiler (DXC) becomes the default shader compiler for Android Vulkan. In addition, DXC support for OpenGL ES has been added.
- Improved the mobile deferred renderer. Improved stability and performance of some modules, including IBL, deferred decals, IES lighting, and other lighting features to match PC-level quality. In addition, fewer shader permutations are used.

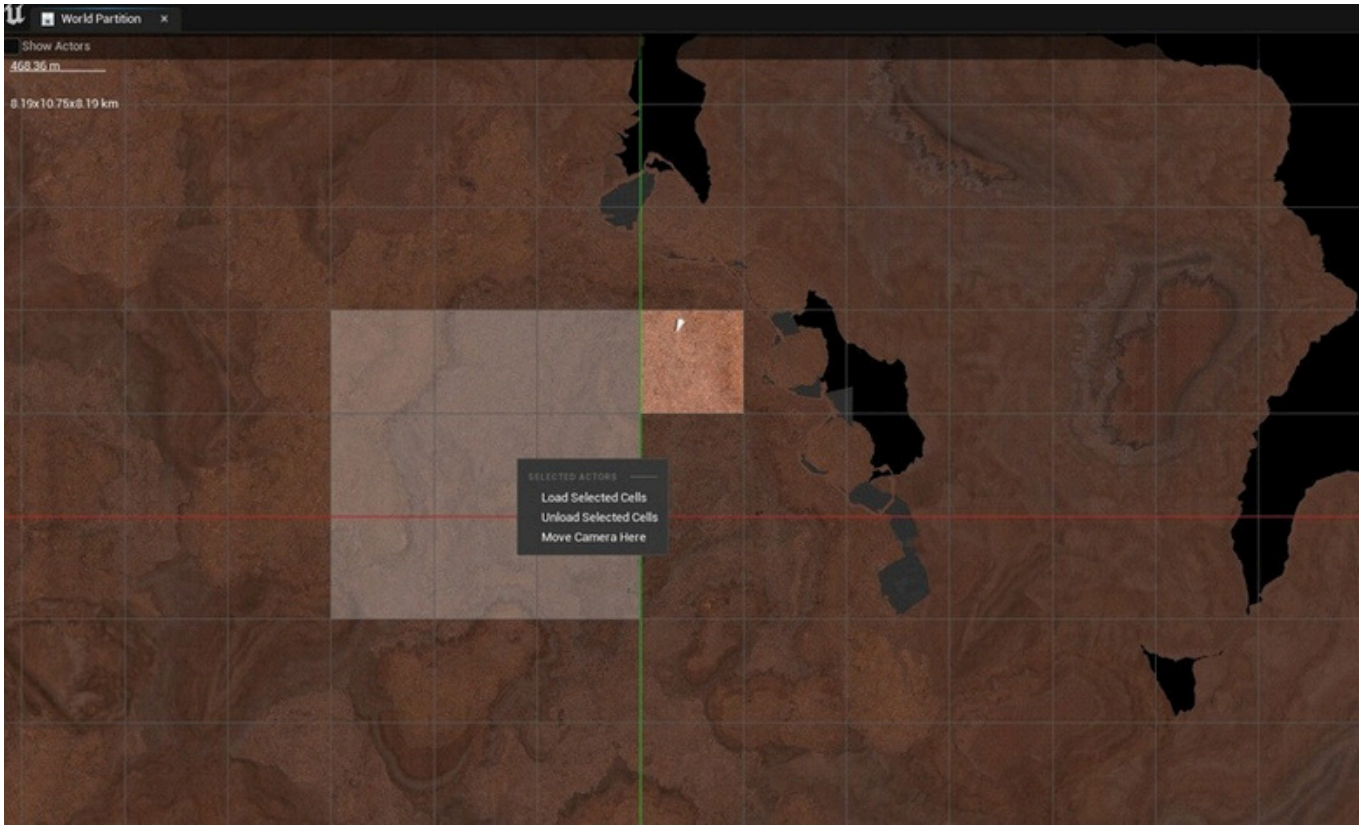
6.2.3 Other new features

6.2.3.1 World Regions

World Partitionis a new data management and streaming system, available both in the editor and at runtime, that completely eliminates the requirement to manually divide your world into countless sub-levels to manage streaming data and reduce data contention.

With world partitioning, the world exists as a single persistent level. In the editor, the world is split into grids, and map data can be loaded partially based on the area of interest. When cooking or launching PIE, the world is divided into grid cells optimized for runtime streaming, becoming independent streaming levels.

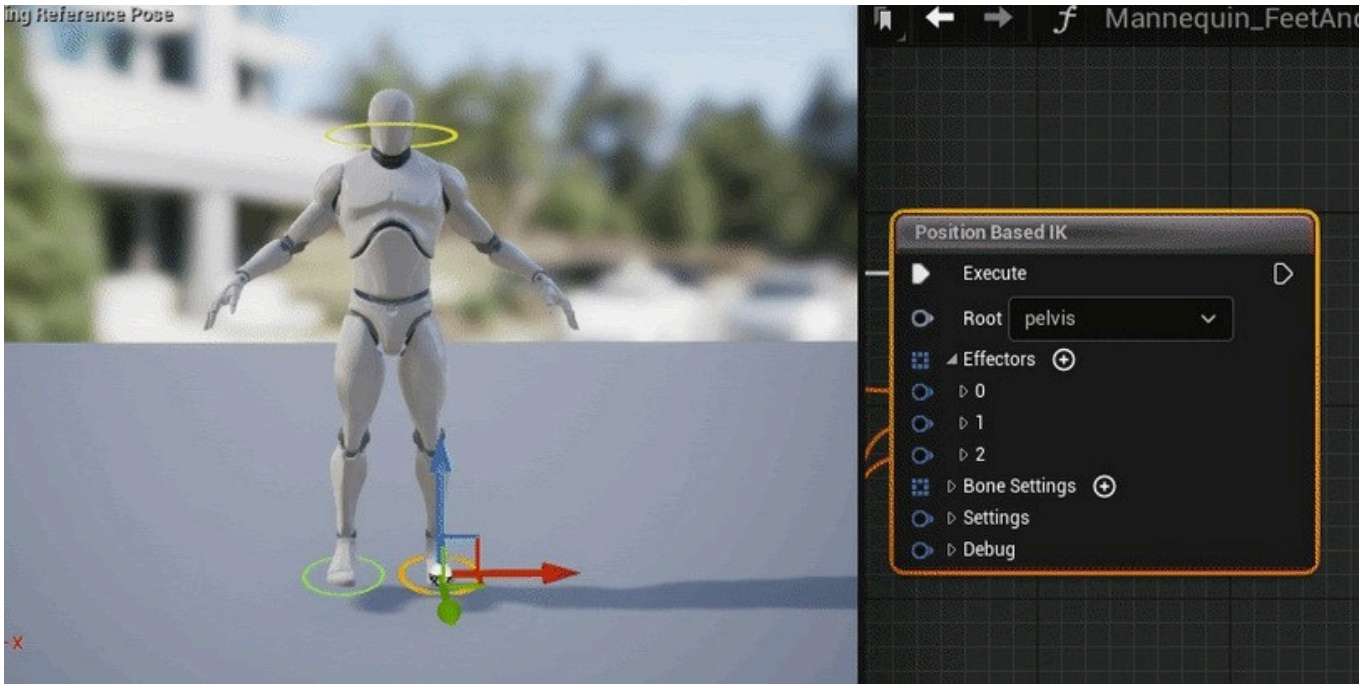
You can open the world partition editor in the menu bar Window/World Partition, drag the left mouse button to quickly select all grids in the specified range, and right-click to pop up the operation menu, including loading the selected grid, unloading the selected grid, and moving the camera to the current grid:



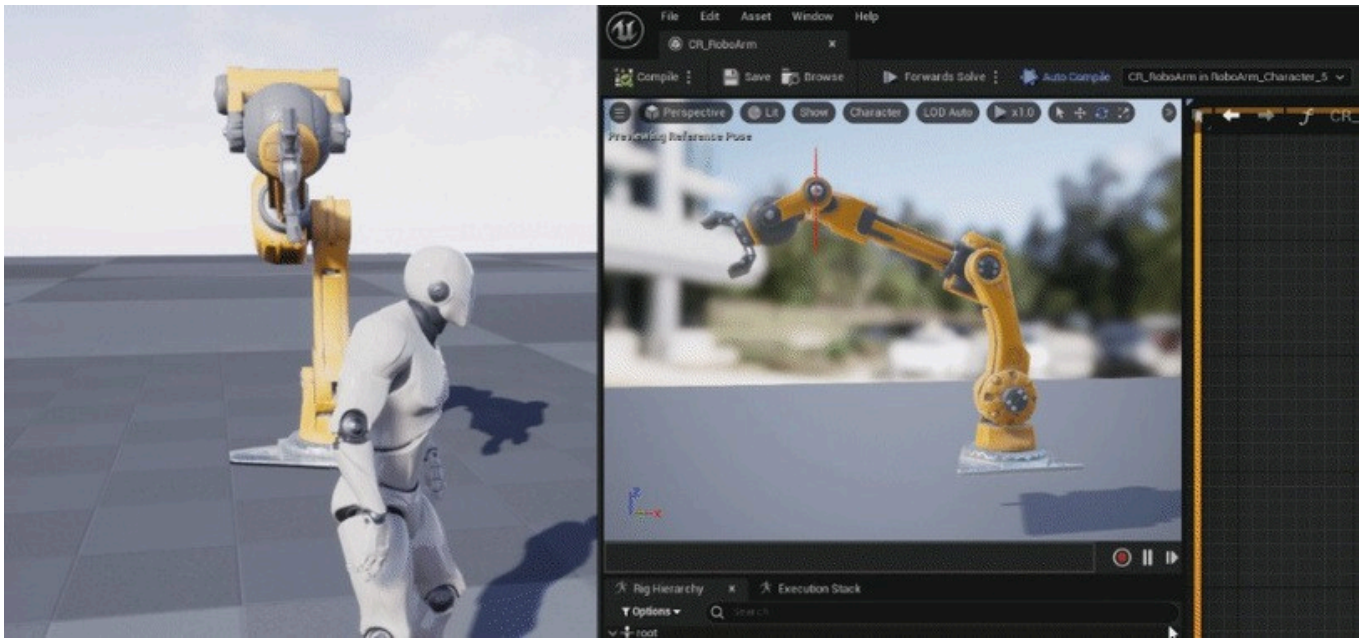
In addition, the world also supports features such as Data Layer, HLOD (Hierarchical Level of Detail), Level Instancing, and One File Per Actor.

6.2.3.2 Animation

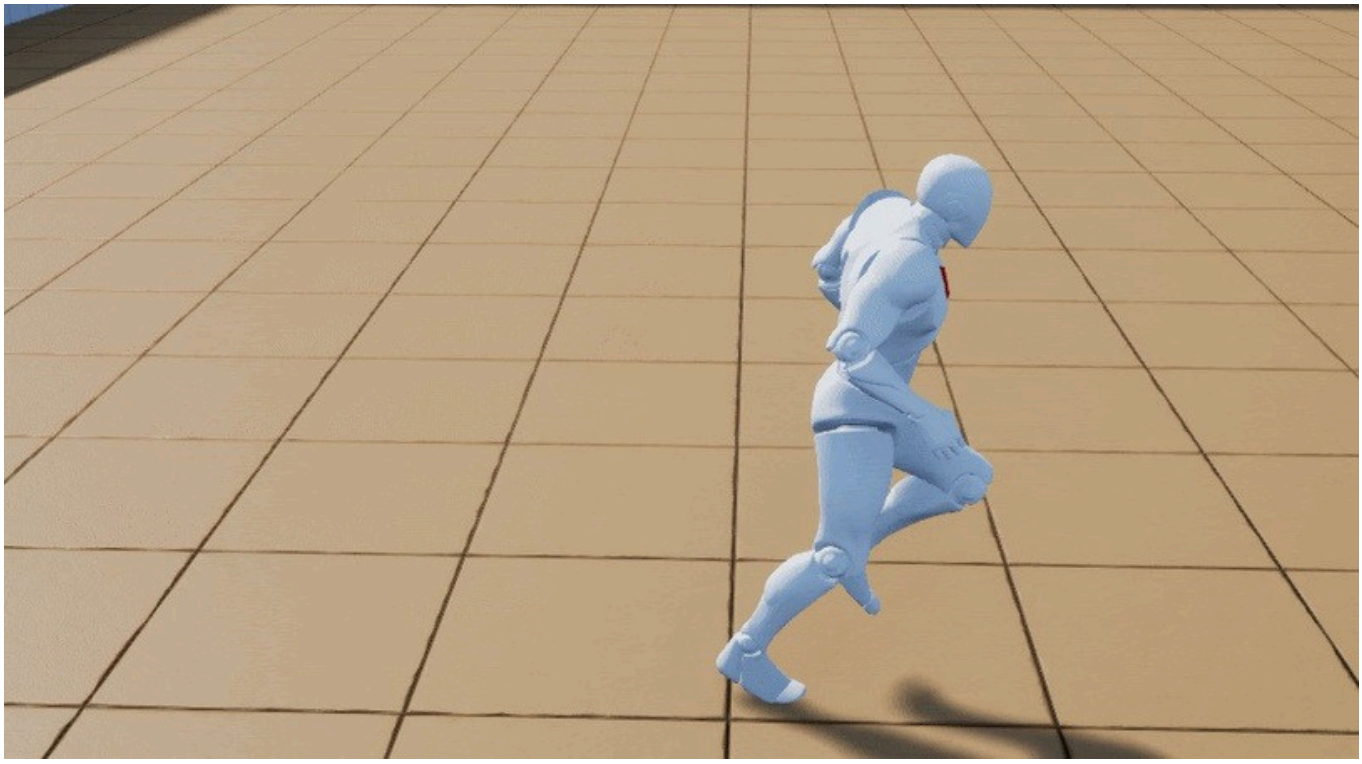
UE5's animation module adds Full Body IK, Control Rig, Motion Warping, animation tool scripts, and support for Sequencer.



Full Body IK diagram.



Schematic diagram of the Control Rig effect.



Schematic diagram of Motion Warping effect.

6.2.3.3 Physics

UE5's physical effects are also very eye-catching among the new features, mainly including the following features:

Chaos

-

Chaos is a lightweight physics simulation solution for UE5, built from the ground up to meet the needs of next-generation games. The main features it supports are:

1. Rigid Body Dynamics

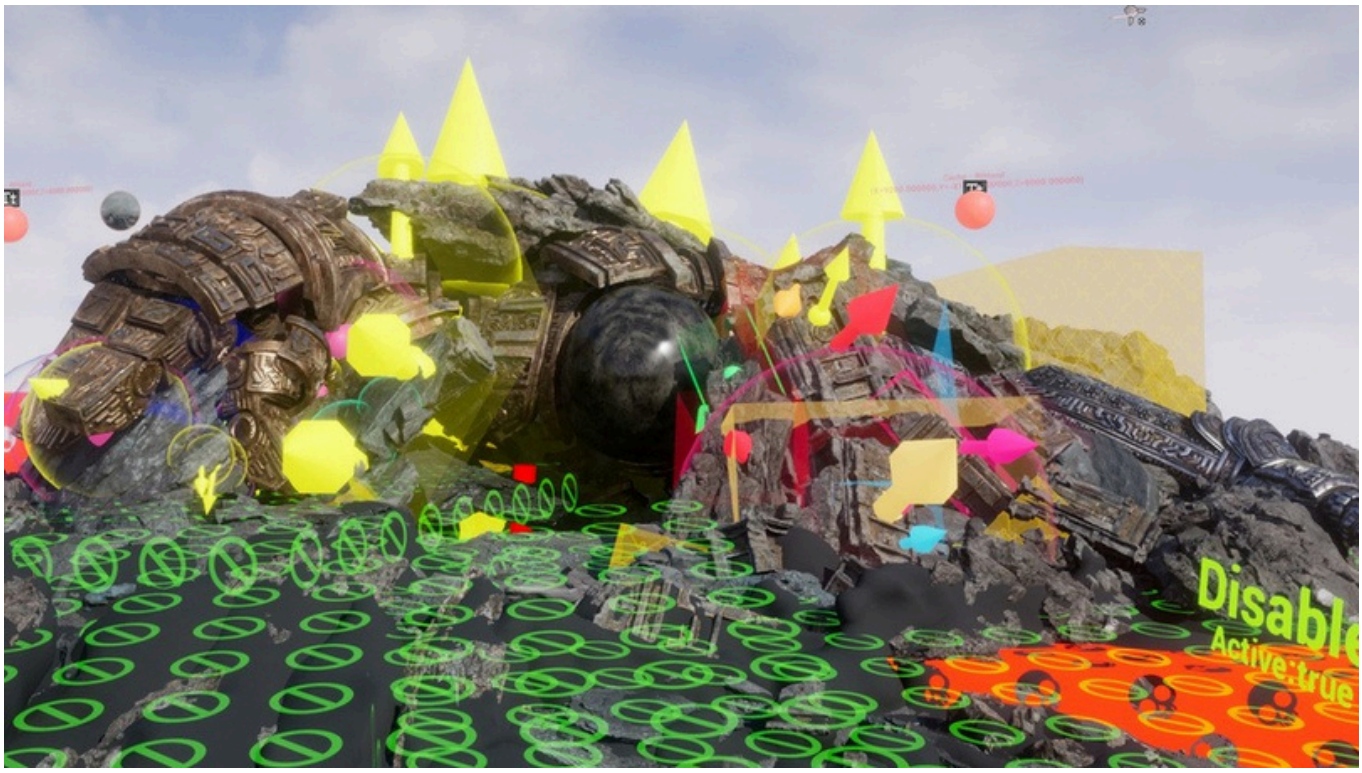
2. Rigid Body Animation Nodes and Cloth Physics

3. Destruction

4. Ragdoll Physics

5. Vehicles

6, physical field (Physics Fields). The physics system allows users to directly affect theChaosPhysics simulation. These fields can be configured to affect the physics simulation in various ways, such as applying forces to rigid bodies, destroying geometric clusters (Geometry Collection Clusters), anchor or disable the fractured rigid body.

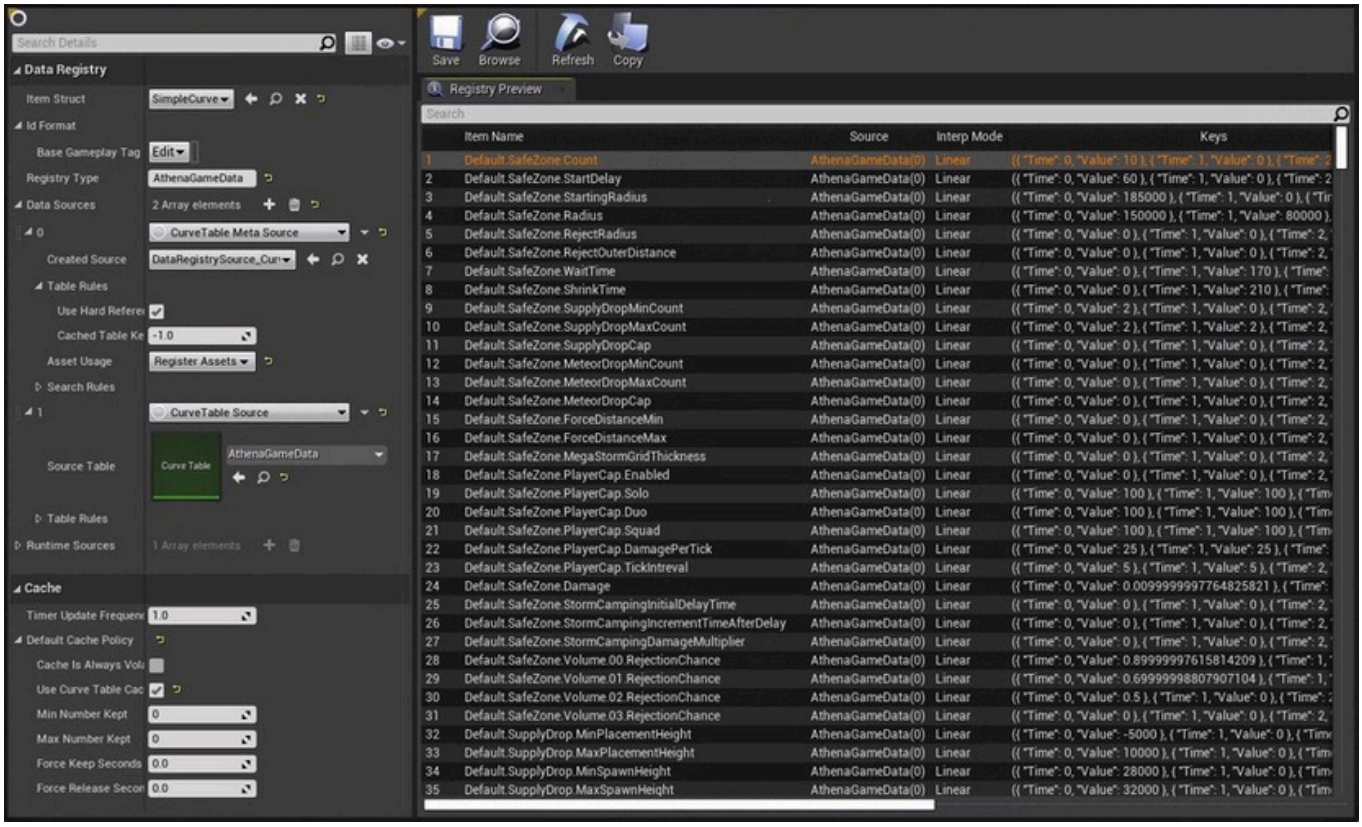


7, fluid simulation (Fluid Simulation)

8, Hair Simulation (Hair Simulation)

6.2.3.4 GamePlay

GamePlayThe framework adds game and module logic, data registry (Data Registries), increased input system.



Data Registry Editor.

6.2.3.5Performance and Platform Management

- **Memory Analysis Tools for Unreal Insights**

UE5passUnreal Insightsof**Memory Insights**The module has improved memory tracking and debugging support. It supports the following features:

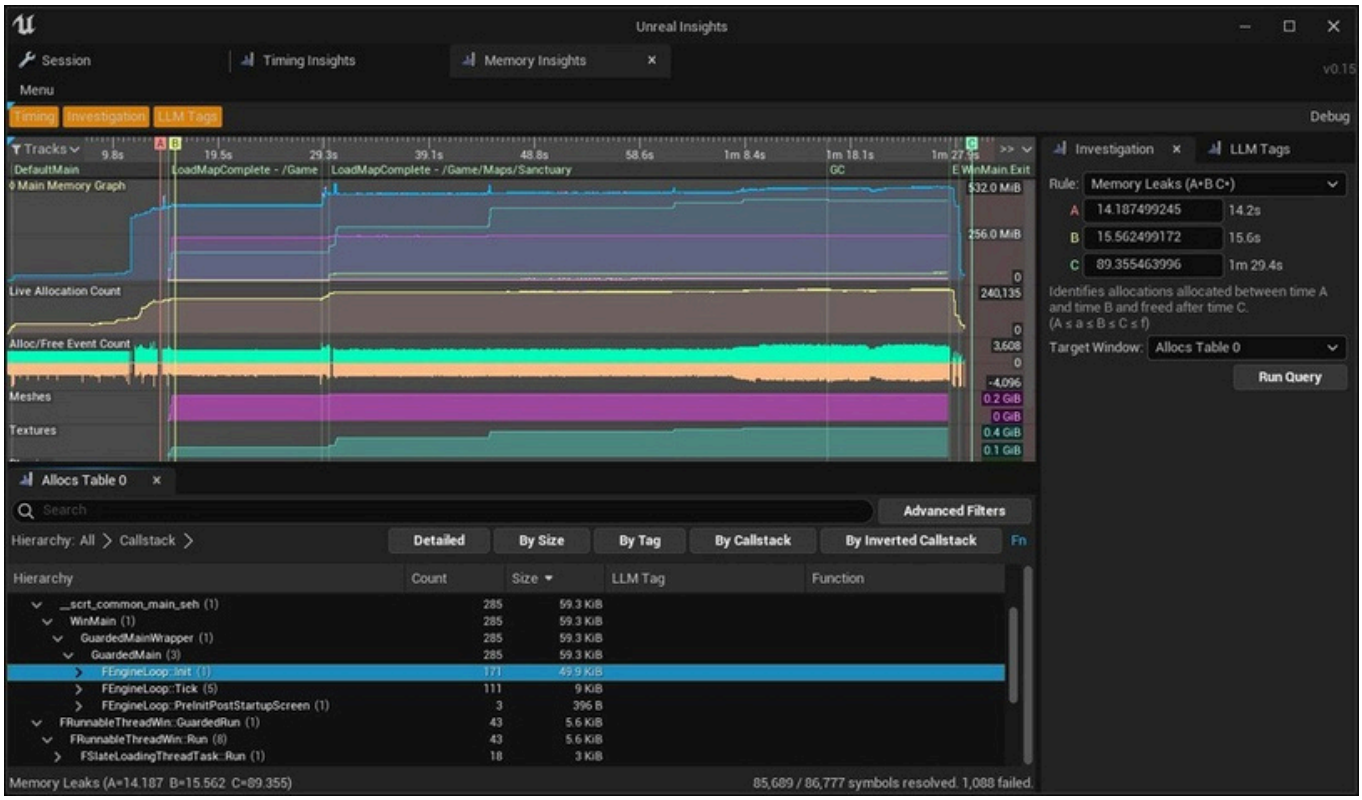
1, view a snapshot of all allocated memory at any given time during a session.

2, compare snapshots of all allocated memory at two different times.

3, view the call stack for each memory allocation.

4, identify long-term and short-term (or temporary) memory allocations.

5, find memory leaks.



*Unreal Insights*of*Memory Insights* Editor overview.

- **Platform Tool Chain**

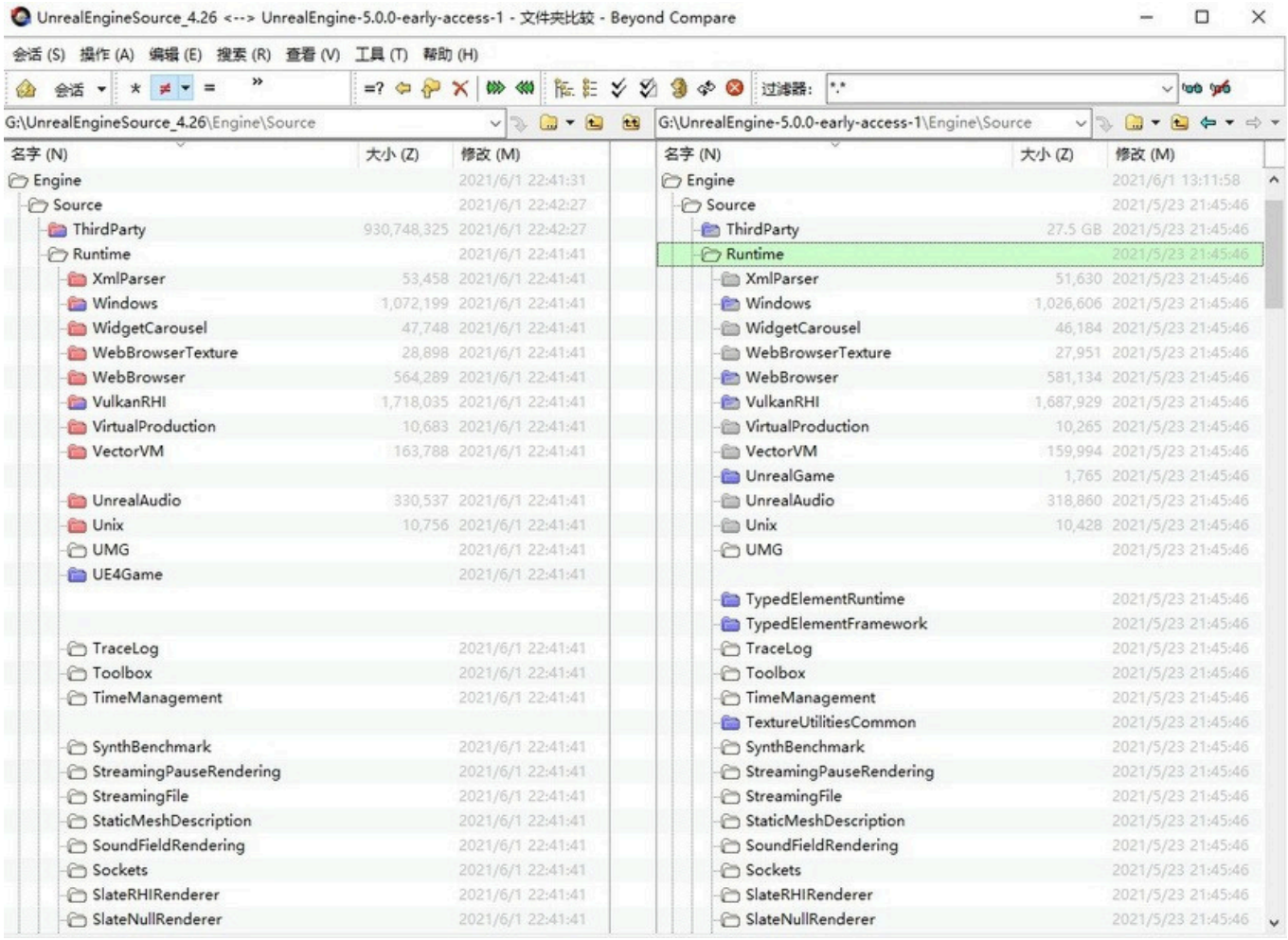
ImprovedMacsuperioriOSThe reliability of the remote build process has been improved, and a cross-platform library has been added to improve the reliability of the remote build process.USBandiOSReliability of device interactions.

In addition, there are added audio effects, redesignedVRtemplate,Unreal Turnkeyetc.

6.3 UE5Rendering system changes

This chapter will compareUE4.26The source code of the version is systematically explained**UE5 EA**Version source code and**UE4.26**For ease of comparison, useBeyond

CompareFolder comparison function:



The source code of the two versions is quite different, but the following chapters of this chapter will focus on the basic modules, rendering system,ShaderThe differences are mainly concentrated in the following folders:

- Engine\Source\Runtime\Core
- Engine\Source\Runtime\CoreUObject
- Engine\Source\Runtime\D3D12RHI
- Engine\Source\Runtime\Engine
- Engine\Source\Runtime\MeshDescription
- Engine\Source\Runtime\Renderer
- Engine\Source\Runtime\RHI
- Engine\Source\Runtime\RHICore
- Engine\Shaders

6.3.1 CoreandCoreUObject

- Core:
    - Modifications and additionsAlgo,IO,Async,Hash,Memory,Math,Container,String,Serialization, UObjectAnd other basic modules.
    - ReviseFileCache,HAL,Logging,Misc,Modules,Statsetc. Support modules.Android,
    - Apple,iOS,HoloLens,Mac,Unix,WindowsAnd other system modules.

- UObject:

    - A lot of revisions and improvementsUObjectThe various attribute modules includeClass,Obj,Package,Propertyand associated modules.
    - IncreaseObjectHandle,ObjectPathId,PackageResourceManager,PropertyClassPrt, PropertyObjectPr tAnd so on modules.
- AssetRegistry:
    - ModifiedAssetDataPart of the logic.
    - improveAssetBundleRelated logic, addAssetDataTagMapConcept.
    - IncreaseAssetDataTagMapSerializationDetails.
- Package:
    - Add or improvePackageNameSome interfaces.
    - IncreasePackageAccessTracking,PackageAccessTrackingOps,PackagePathAnd other module logic.
- Serialization:
    - Modified serializationBulkData,ArchiveUObject,JsonBasic modules such as reading and writing. A lot of modifications have been made to
    - modules such as loading, asynchronous loading, and writing documents.
    - IncreaseFilePackageStore,MappedNameAnd other modules.

It can be seen that a lot of changes and reconstructions have been made in the core and basic modules, involving all aspects.

6.3.2 RHIandRHICore

- RHI:
    - Added markup and support for some rendering features, including several features for mobile devices.
    - IncreaseMSAA,BufferUsageFlags,RHIPipeline,EResourceTransitionFlags,TransitionAnd other concepts and interfaces.
    - deleteFRHIVertexBuffer,FRHIIndexBufferandFRHIStructuredBufferType, unifiedFRHIBuffer, modify the relevant interfaces.
    - IncreaseTexture2DArrayRelated interfaces, delete the oldRHIAll kinds of resource creation interfaces, unified use
    - RHICreateImprove ray tracing types and interfaces, such asRayTraceDispatchIndirect,IsRayTracingShaderFrequency
    - wait. GlobalUniformBufferChanged toStaticUniformBuffer. ERHIFeatureLevelIncreaseSM6, improve the platform-
    - related detection or data interface. AddPCD3D_SM6ofShaderPlatform, delete partShaderPlatform,likePS4,
    - XBOXONE_D3D12,OPENGL_ES31_EXTwait. FGenericDataDrivenShaderPlatformInfoAdded
    - many tags and corresponding acquisition interfaces.EShaderCodeResourceBindingType,
    - EUniformBufferBindingFlags, FRHITextureCreateInfo,ERHITextureMetaDataAccess, ERHITextureSRVOverrideSRGBType,FRHITextureSRVCreateInfo, FRHITextureUAVCreateInfo,FRHIBufferCreateInfo,FRHIBufferSRVCreateInfo, FRHIBufferUAVCreateInfoetc. types and interfaces.
    - CompleteFRHIGraphicsPipelineState,FRHITextureReference,RHIValidationetc.

    - IncreaseRHITransientResourceAllocator,RHIValidationTransientResourceAllocatorAnd other resource conversion modules.
    - Modify or addFTransientState,FGlobalUniformBuffers,FPipelineStateAnd other types.

- RHICore:
    - New modules includeRHICoreShader,RHICoreTransientResourceAllocator, RHIPoolAllocatorAnd other modules.

6.3.3 Renderer

- Add instantiation clipping module:InstanceCulling,InstanceCullingManageretc., including FInstanceCullingRdgParams,EInstanceCullingMode,FInstanceCullingContext, FInstanceCullingManageretc., mainly used forNanitetechnology.
- Virtual textures increase or improve data reading and writingFVirtualTextureFeedbackBuffer, RenderPages, RenderPagesStandAloneetc.
- Global distance field data (FGlobalDistanceFieldParameterData) has increasedMipmapandVTData and interfaces.
- HairStrandIncreaseEHairBindingType,EHairInterpolationType,FHairStrandsInstanceAnd other types.
- Add or improveFVertexFactoryShaderPermutationParametersType.
- MeshPassProcessor:
    - EMeshPassIncreaseVSMShadowDepth,LumenCardCapture,EditorLevelInstanceetc. dedicated channels.
    - IncreaseEFVisibleMeshDrawCommandFlags,FCompareFMeshDrawCommands, FMeshPassProcessorRenderStateAnd other types.
    - IncreaseFSimpleMeshDrawCommandPassType, used to handle places where drawing commands do not need to be processed in parallel to reduce overhead.
    - FVisibleMeshDrawCommandIncreaseEFVisibleMeshDrawCommandFlagsAnd forGPUCullof FMeshDrawCommandSor tKey,InRunArraywait.

- FDynamicPassMeshDrawListContextofFinalizeCommandThe stage has increased NewVisibleMeshDrawCommand.Setupstage.
- AbandonSetInstancedViewUniformBuffer,SetPassUniformBuffer,GetInstancedViewUniformBuffer wait UniformBufferInterface.
- AddedNaniteModules:
    - AddedNaniteRender:FNaniteCommandInfo,ENaniteMeshPass, FNaniteDrawListContext, FCullingContext,FRasterContext,FRasterResults, FNaniteShader,FNaniteMaterialVS, FNaniteMeshProcessor,FNaniteMaterialTables, ERasterTechnique,ERasterScheduling, EOutputBufferMode,FPackedViewAnd other types and processing interfaces.

    - FPrimitiveFlagsCompactIncreasebIsNaniteMeshmark.
    - FPrimitiveSceneInfoIncreaseNaniteCommandInfos,NaniteMaterialIds,LumenPrimitiveIndex as well as CachedRayTracingMeshCommandsHashPerLOD,bRegisteredWithVelocityData, InstanceDataOffset, NumInstanceDataEntriesInstantiation and ray tracing related data and processing interfaces.

- AddedLumenModules:
    - There are many modules involved, which can be summarized as follows:DiffuseIndirect,Scene,HardwareRayTracing, Mesh, Probe,Radiance,Radiosity,TranslucencyVolume,VoxelAs well as data structures, toolboxes, etc.
- Replace traditionalShaderBind interface toRDG,likeSHADER_PARAMETER_TEXTURE Change to SHADER_PARAMETER_RDG_TEXTURE.
- Add or improveRuntimeVirtualTextureProducer,FSceneTextureExtracts, EMobileSceneTextureSetupMode,StrataEnhanced post-processing effects, such as addingTemporal Super Resolution（TSR).
- Enhanced light tracing modules, such asRTGI,RTAO,RTR,RTShadow,RTSkyLightwait.
- DeferredShadingRenderer:
    - IncreaseLumen,IndirectLightRender,SSRT,TranslucencyLightingVolumeAnd other related modules, types and steps.
    - IncreaseFLumenCardRenderer,TPipelineState,EDiffuseIndirectMethod, EAmbientOcclusionMethod,EReflectionsMethod,FPerViewPipelineState, FFamilyPipelineStateAnd other types.
- Increased or enhancedBasePass,DepthPass,SDF,GPUScene,GlobalDistanceField,BVH, GenerateConser vativeDepthBuffer,Light Rendering,InrectLightRendering, MeshDrawCommands ,Shader,ScreenSpace,Shadow,MobileRenderWait for the rendering module.

fromUE4.26arrive5.0EA, important and basic rendering modules have undergone major and minor modifications.

6.3.4 Engine

- AddedIRenderCaptureProvider,NaniteResources,NaniteStreamingManageretc.LevelInstance
- Module, used to handle data reading and writing, packaging, rendering, editor, etc. of level
- instantiation.ActorReferencesUtils,AssetCompilingManager,AsyncCompilationHelpers, CanvasRender,ComputeKernelCollection,DerivedMeshDataTaskUtils, InstancedStaticMeshDelegates,InstanceUniformShaderParameters, MeshCardRepresentation, NaniteSceneProxy,ObjectCacheContext,StaticMeshCompiler, TextureCompilerModules such as Buffer,Mesh,Nanitewait. PrimitiveSceneProxy:
    - IncreaseLevelInstance,bLevelInstanceEditingState,TranslucencySortDistanceOffset, MeshCardRepresentationAnd other data and interfaces.

    - IncreaseSupportsNaniteRendering,IsNaniteMesh,bSuppor tsMeshCardRepresentation, IsAlwaysVisible,GetPrimitiveInstances,RayTracingGroupId, RayTracingGroupCullingPriorityAnd other data and interfaces.
- SceneInterfaceandScene Management:
    - IncreaseFInstanceCullingManagerResourcesetc., useFGPUScenePrimitiveCollectorreplace FPrimitiveUniformShaderParameters.
    - IncreaseHairStrand,PhysicsField,LumenSceneCard,ComputeFrameworkOther operation interfaces.
- SceneView:
    - IncreaseComputeNearPlane,GetScreenPositionScaleBiasetc.FViewShaderParameters,
    - GeneralPurposeTweak2, PrecomputedIndirectLightingColorScale,GlobalDistanceField ,VirtualTexture, PhysicsField,Lumen,Instance,PagewaitshaderBinding.
- Enhanced distance field,GPUSkinCache,Material,StaticMesh,SkeletalMesh,Textureetc. types and interfaces.

6.3.5 Shaders

- AddedNanite:
    - AddedClusterCulling,Culling,HZBCull,InstanceCulling,MaterialCulling,Shadow, GBuffer, Imposter,DataDecode,DataPacked,Rasterizer,WritePixelwait.
- AddedLumen:
    - andC++Similar, there are many modules involved, which can be summarized as follows:DiffuseIndirect,Scene, HardwareRayTracing,Mesh,Probe,Radiance,Radiosity,TranslucencyVolume,Voxel As well as data structures, toolboxes, etc.
    - AddedFinalGather,deal withLumenProbeofHierarchy,Occlusionas well asIndex,Samplewait.
- AddedStrata:
    - DeferredLighting,EnvironmentLighting,Evaluation,ForwardLighting,Material, and other modules.
- AddedVirtualShadowMap:
    - Added a module for constructing per-page drawing commands.
    - Added cache management.
    - AddedPage,Projection,SMRTAnd other modules.
- AddedInstanceCulling:
    - BuildInstanceDrawCommandsModule, providingGPUSceneexistCompute ShaderDynamically crop and generate drawing instructions.
    - CullInstancesModule, inCompute ShaderPrimitives that are out of view or obscured are clipped.
    - InstanceCullingCommonModules define basic types and interfaces.

- EnhancedHairStrandsModules such asHairCards,HairScatter,BsdfPlot,ClusterCulling,Shadow, DeepShadow,EnvironmentLighting,GBuffer,Material,Visibility,VoxelAnd other modules.
  EnhancementPathTracing,RayTracing,SSD,SSRT,TAAAnd other modules.
-
- DeletedLPVmodule.
- Improve the foundation, materials, colors, lighting, shadows, and specialPassModules such as AnisotropyPass,BasePass, MobileBasePass,BRDF,BurleySSS,CapsuleLigh,RectLight, TranslucentLighting, ClusteredDeferredShading,LightGrid,ForwardLighting,DeferredLight, DiffuseIndirect, DistanceFieldAO,DistanceFieldLighting,DistanceFieldShadow,GlobalDistanceField, Math, Decal,GpuSkin,Halton,MonteCarlo,HkDJ,LocalVertexFactor y,MaterialTemplate, Particle, PlanarReflection,PostProcess,Reflection,SceneData,ShadingCommon, ShadingModels,Shadow, Volumetricwait.

6.3.6 UE5Rendering system summary

As can be seen from the previous sections, the biggest change is the addition ofNanite,Lumen,VSM,InstanceCull,LevelInstance Modules and technologies, and modifiedEngine,RenderModule-related types and interfaces.

RHIThe changes in the layers are mainly to change the various vertices and indicesBufferUnifiedFRHIBuffer.RendererThe layer enhances ray tracing, especially the screen

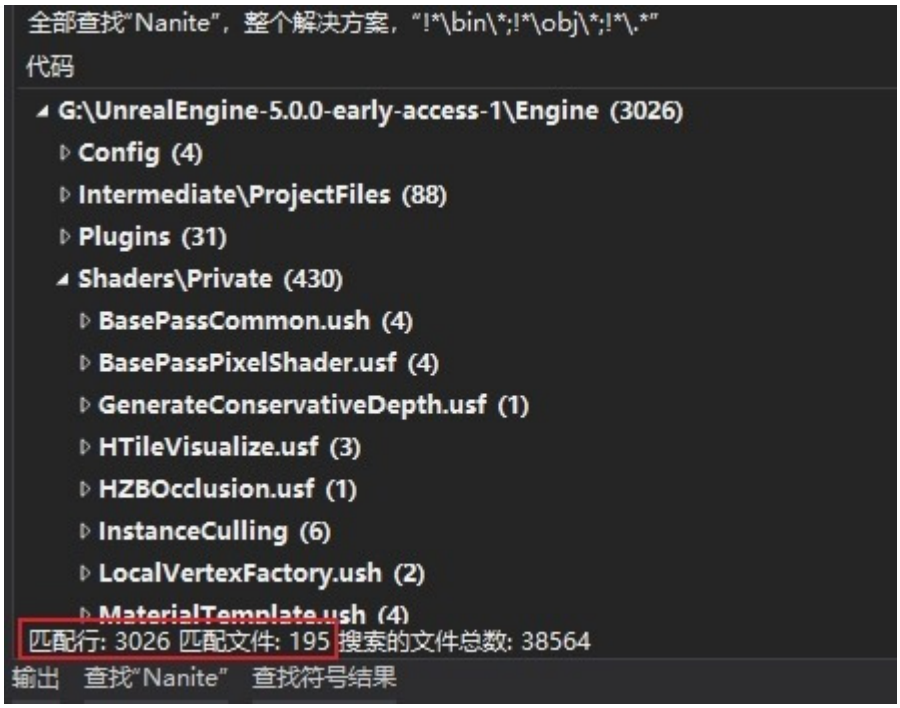
Screen space ray tracing enhances the various applications of distance fields and removesLPV.EngineThe layer mainly revolves aroundRHI,RendererLayer Change

The actions were modified and adjusted accordingly.

6.4 Nanite

This chapter will explainUE5ofNanitePreprocessing, rendering and optimization techniques of virtual micropolygons.

existUE5 EASource code engineering search"Nanite"Words, found195Files for3026Matches:



Since the scope is too broad, it is impossible to explain every detail. After screening, the author will focus on analyzing the following modules:NaniteSource code:

- EditorofNanite MeshBuild process.
- EngineModule AboutNaniteResource management, loading, assembly, etc.
- RendererModule AboutNaniteRendering process and optimization technology.
- ShadermiddleNaniteRendering steps and algorithms.

6.4.1 NaniteBase

This section mainly discussesNaniteRelated basic concepts, types and fundamental knowledge.

6.4.1.1 FMeshNaniteSettings

```
// Engine\Source\Runtime\Engine\Classes\Engine\EngineTypes.h

//Shadow map method.
namespace   EShadowMapMethod
{
    enum Type
    {
        //Traditional shadow maps. Component-by-component culling, resulting in poor performance in
        high-polygon scenes. ShadowMapsUMETA(DisplayName ="Shadow Maps"),
        //Renders geometry into virtual depth maps for shadows, providing high quality next-generation projection with a simple setup.Nanite Now
                                                                                                When used together,
efficient cutting.
        VirtualShadowMapsUMETA(DisplayName ="Virtual Shadow Maps (Beta)")
    };
}

//ApplicationNaniteConfiguration when building data.
struct    FMeshNaniteSettings
{         Is it enabled?NaniteGrid.
    //
    uint8 bEnabled :1;
    //Position accuracy. The step length is2^(-PositionPrecision) cm. MIN_int32Indicates automatic setting.
```

```
    int32 PositionPrecision;
    //fromLOD0The triangle percentage.1.0It means there is no reduction in size.0.0Indicates that there is no triangle.
    floatPercentTriangles;

    FMeshNaniteSettings(): bEnabled(false), PositionPrecision(MIN_int32), PercentTriangles(0.0f){}

    FMeshNaniteSettings(constFMeshNaniteSettings& Other);

    booloperator==(constFMeshNaniteSettings& Other)const; booloperator!=(
    constFMeshNaniteSettings& Other)const;
};
```

6.4.1.2 StaticMesh

```
// Engine\Source\Runtime\Engine\Classes\Engine\StaticMesh.h

classUStaticMesh: public UStreamableRenderAsset, (.....) {

    (.....)

public:
    //Static GridNaniteConfiguration data.
    FMeshNaniteSettings NaniteSettings;

    //If the grid has a validNaniteRendering data
    returnstrue. bool HasValidNaniteData()const {

        if(constFStaticMeshRenderData* SMRenderData = GetRenderData()) {

            returnSMRenderData->NaniteResources.PageStreamingStates.Num() >0;
        }
        return                  false;
    }

    (.....)

    //Ultra-high resolution source model related interfaces.
    FStaticMeshSourceModel&GetHiResSourceModel();
    constFStaticMeshSourceModel&GetHiResSourceModel()const;
    FStaticMeshSourceModel&&MoveHiResSourceModel();
    void SetHiResSourceModel(FStaticMeshSourceModel&& SourceModel);


    bool LoadHiResMeshDescription(FMeshDescription& OutMeshDescription)const; bool
    CloneHiResMeshDescription(FMeshDescription& OutMeshDescription)const; FMeshDescription*
                            CreateHiResMeshDescription();
    FMeshDescription*       CreateHiResMeshDescription(FMeshDescription MeshDescription);
    FMeshDescription*       GetHiResMeshDescription()const;
    bool IsHiResMeshDescriptionValid()              const;
    void CommitHiResMeshDescription(constFCommitMeshDescriptionParams& Params); void
    ClearHiResMeshDescription();

    (.....)

private:
    //Ultra-high resolution source model.
    FStaticMeshSourceModel      HiResSourceModel;
```

```
    (......)
};


// Engine\Source\Runtime\Engine\Public\StaticMeshResources.h

//Rendering data required for static meshes.
classFStaticMeshRenderData {

public:
    (......)

    // NaniteRendering resources.
    Nanite::FResources NaniteResources;


    (......)
};
```

6.4.1.3 NaniteResource

```
// Engine\Source\Runtime\Engine\Public\Rendering\NaniteResources.h

//Maximum number of constants.
#defineMAX_STREAMING_REQUESTS                    ( 128u * 1024u ) 128
#define    MAX_CLUSTER_TRIANGLES
#define    MAX_CLUSTER_VERTICES                  256
#defineMAX_CLUSTER_INDICES                       ( MAX_CLUSTER_TRIANGLES * 3 )
#define    MAX_NANITE_UVS                        4
#define    NUM_ROOT_PAGES                        1u

//Whether to use triangle band indexing.
#defineUSE_STRIP_INDICES                         1

//          constanCt.LUSTER                     #define
CLUSTER_PAGE_GPU_SIZE_BITS                       #define    17
CLUSTER_PAGE_GPU_SIZE                            #define    ( 1 << CLUSTER_PAGE_GPU_SIZE_BITS )
CLUSTER_PAGE_DISK_SIZE                           #define    ( CLUSTER_PAGE_GPU_SIZE * 2 )
MAX_CLUSTERS_PER_PAGE_BITS                       #define    10
MAX_CLUSTERS_PER_PAGE_MASK 1 ) #define            ( ( 1 << MAX_CLUSTERS_PER_PAGE_BITS ) -
MAX_CLUSTERS_PER_PAGE                            #define
MAX_CLUSTERS_PER_GROUP_BITS                      #define    ( 1 << MAX_CLUSTERS_PER_PAGE_BITS )
MAX_CLUSTERS_PER_GROUP_MASK      - 1 )           9
#defineMAX_CLUSTERS_PER_GROUP 1 )                ( ( 1 << MAX_CLUSTERS_PER_GROUP_BITS )
#defineMAX_CLUSTERS_PER_GROUP_TARGET

                                                 ( ( 1 << MAX_CLUSTERS_PER_GROUP_BITS ) -

                                                 128

//Level,    GPUConstants for pages, instantiation groups, etc.
#define    MAX_HIERACHY_CHILDREN_BITS            6
#defineMAX_HIERACHY_CHILDREN                     ( 1 << MAX_HIERACHY_CHILDREN_BITS ) 14
#define    MAX_GPU_PAGES_BITS
#define        MAX_GPU_PAGES                         ( 1 << MAX_GPU_PAGES_BITS )
#define    MAX_INSTANCES_BITS                    twenty four
#defineMAX_INSTANCES                             ( 1 << MAX_INSTANCES_BITS ) 16
#define    MAX_NODES_PER_PRIMITIVE_BITS
#define    MAX_RESOURCE_PAGES_BITS               20
```

```
#define          MAX_RESOURCE_PAGES                    (1 << MAX_RESOURCE_PAGES_BITS)
#define       MAX_GROUP_PARTS_BITS                  3
#define       MAX_GROUP_PARTS_MASK                  ((1 << MAX_GROUP_PARTS_BITS) - 1) (1 <<
#define    MAX_GROUP_PARTS                          MAX_GROUP_PARTS_BITS)

#definePERSISTENT_CLUSTER_CULLING_GROUP_SIZE        64

//BVH
#defin     MAX_BVH_NODE_FANOUT_BITS                 3
e          MAX_BVH_NODE_FANOUT                      (1 << MAX_BVH_NODE_FANOUT_BITS)
#defin     MAX_BVH_NODES_PER_GROUP                  (PERSISTENT_CLUSTER_CULLING_GROUP_SIZE
eMAX_BVH_NODE_FANOUT)
#defin
#define    NUM_CULLING_FLAG_BITS                    3
#define    NUM_PACKED_CLUSTER_FLOAT4S               8
#defineMAX_POSITION_QUANTIZATION_BITS               twenty one // (21*3 = 63) < 64
#defineNORMAL_QUANTIZATION_BITS                     9

#define    MAX_TEXCOORD_QUANTIZATION_BITS           15
#define    MAX_COLOR_QUANTIZATION_BITS              8

#define    NUM_STREAMING_PRIORITY_CATEGORY_BITS     2
#define    STREAMING_PRIORITY_CATEGORY_MASK         ((1u <<
NUM_STREAMING_PRIORITY_CATEGORY_BITS)      - 1u)

#define    VIEW_FLAG_HZBTEST                        0x1

#define    MAX_TRANSCODE_GROUPS_PER_PAGE            128

#define       VERTEX_COLOR_MODE_WHITE               0
#define       VERTEX_COLOR_MODE_CONSTANT            1
#define       VERTEX_COLOR_MODE_VARIABLE            2

#define    NANITE_USE_SCRATCH_BUFFERS               1

#define    NANITE_CLUSTER_FLAG_LEAF                 0x1


namespace Nanite
{

//Integer vector.
structFUIntVector
{
    uint32 X, Y, Z;

    booloperator==(constFUIntVector& V)const;
    FORCEINLINE friend FArchive& operator<<(FArchive& Ar, FUIntVector& V);
};
//Packed level nodes.
structFPackedHierarchyNode {



    FSphere             LODBounds[MAX_BVH_NODE_FANOUT];    //  Made of spheresLODBounding box.

    struct
    {
        FVector             BoxBoundsCenter;
```

```
        uint32          MinLODError_MaxParentLODError;
    } Misc0[MAX_BVH_NODE_FANOUT];

    struct
    {
        FVector         BoxBoundsExtent;
        uint32          ChildStartReference;
    } Misc1[MAX_BVH_NODE_FANOUT];

    struct
    {
        uint32          ResourcePageIndex_NumPages_GroupPartSize;
    } Misc2[MAX_BVH_NODE_FANOUT];
};

//Material triangle.
structFMaterialTriangle {

    uint32  Index0;     uint32
    Index1;             uint32
    Index2;             uint32
    MaterialIndex;      uint32
    RangeCount;
};

//fromValueGets the value of the specified number of bits and offset.
uint32GetBits(uint32 Value, uint32 NumBits, uint32 Offset) {

    uint32 Mask = (1u<< NumBits) -1u; return(Value
    >> Offset) & Mask;
}
//Merges the value of the specified number of bits and offset intoValuemiddle.
void SetBits(uint32& Value, uint32 Bits, uint32 NumBits, uint32 Offset) {

    uint32 Mask = (1u<< NumBits) -1u; Mask <<=
    Offset;
    Value = (Value & ~Mask) | (Bits << Offset);
}

//quiltGPUUse the packageCluster.
                    struct
                    FPackedCluster
{
    //Data members required for rasterization.
    FIntVector      QuantizedPosStart;
    uint32          NumVerts_PositionOffset;                    //  NumVerts:9,     PositionOffset:23

    FVector         MeshBoundsMin;
    uint32          NumTris_IndexOffset;                        // NumTris: 8, IndexOffset: 24

    FVector         MeshBoundsDelta;
    uint32          BitsPerIndex_QuantizedPosShift_PosBits;         //  BitsPerIndex:4,
QuantizedPosShift:6,        QuantizedPosBits:5.5.5

    //Cut out the required data members.
    FSphere         LODBounds;

    FVector         BoxBoundsCenter;
    uint32          LODErrorAndEdgeLength;
```

```
        FVector         BoxBoundsExtent;
        uint32          Flags;

        //Data members required by the material.
        uint32          AttributeOffset_BitsPerAttribute; 10        //  AttributeOffset: twenty two,
BitsPerAttribute:
        uint32          DecodeInfoOffset_NumUVs_ColorMode;      // DecodeInfoOffset: 22, NumUVs:
3, ColorMode: 2
        uint32          UV_Prec;                                // U0:4, V0:4, U1:4, V1:4, U2:4,
V2:4, U3:4, V3:4
        uint32          PackedMaterialInfo;

        uint32          ColorMin;
        uint32          ColorBits;                              // R:4, G:4, B:4, A:4
        uint32          GroupIndex;                             // Debug only
        uint32          Pad0;

        uint32          GetNumVerts()const                      {return
GetBits(NumVerts_PositionOffset,9,0); }
        uint32          GetPositionOffset()const                { return
GetBits(NumVerts_PositionOffset,twenty three,9); }

        uint32          GetNumTris()const                       { return
 GetBits(NumTris_IndexOffset,8,0); }
        uint32          GetIndexOffset()const                   { return
GetBits(NumTris_IndexOffset,twenty four,8); }

        uint32          GetBitsPerIndex()const                  {return
GetBits(BitsPerIndex_QuantizedPosShift_PosBits,4,0); }
        uint32          GetQuantizedPosShift()const             { return
GetBits(BitsPerIndex_QuantizedPosShift_PosBits,6,4); }
        uint32          GetPosBitsX()const                      { return
GetBits(BitsPerIndex_QuantizedPosShift_PosBits,5,10); }
        uint32          GetPosBitsY()const                      { return
GetBits(BitsPerIndex_QuantizedPosShift_PosBits,5,15); }
        uint32          GetPosBitsZ()const                      { return
GetBits(BitsPerIndex_QuantizedPosShift_PosBits,5,20); }

        uint32          GetAttributeOffset()const               { return
GetBits(AttributeOffset_BitsPerAttribute,twenty two,0); }
        uint32          GetBitsPerAttribute()const              { return
GetBits(AttributeOffset_BitsPerAttribute,10,twenty two); }

        void            SetNumVerts(uint32 NumVerts)            { SetBits(NumVerts_PositionOffset,
NumVerts,9,0); }
        void            SetPositionOffset(uint32 Offset)        { SetBits(NumVerts_PositionOffset,
Offset,twenty three,9); }

        void            SetNumTris(uint32 NumTris)              { SetBits(NumTris_IndexOffset,
NumTris,8,0); }
        void            SetIndexOffset(uint32 Offset)           { SetBits(NumTris_IndexOffset,
Offset,twenty four,8); }

        void            SetBitsPerIndex(uint32 BitsPerIndex)    {
SetBits(BitsPerIndex_QuantizedPosShift_PosBits, BitsPerIndex,4,0); }
        void            SetQuantizedPosShift(uint32 PosShift) {
SetBits(BitsPerIndex_QuantizedPosShift_PosBits, PosShift,6,4); }
```

```
        void            SetPosBitsX(uint32 NumBits)                                 {
SetBits(BitsPerIndex_QuantizedPosShift_PosBits, NumBits,5,10); }
        void            SetPosBitsY(uint32 NumBits)                                 {
SetBits(BitsPerIndex_QuantizedPosShift_PosBits, NumBits,5,15); }
        void            SetPosBitsZ(uint32 NumBits)                                 {
SetBits(BitsPerIndex_QuantizedPosShift_PosBits, NumBits,5,20); }

        void            SetAttributeOffset(uint32 Offset)                           {
  SetBits(AttributeOffset_BitsPerAttribute, Offset,twenty two,0); }
        void            SetBitsPerAttribute(uint32 Bits)                           {
  SetBits(AttributeOffset_BitsPerAttribute, Bits,10,twenty two); }

        void            SetDecodeInfoOffset(uint32 Offset)                            {
  SetBits(DecodeInfoOffset_NumUVs_ColorMode, Offset,twenty two,0); }
        void            SetNumUVs(uint32 Num)                                      {
SetBits(DecodeInfoOffset_NumUVs_ColorMode, Num,3,twenty two); }
        void            SetColorMode(uint32 Mode)                                  {
SetBits(DecodeInfoOffset_NumUVs_ColorMode, Mode,2,twenty two+3); } };


//Page flow status.
structFPageStreamingState {

        uint32                  BulkOffset;     BulkSize;
        uint32                  PageUncompressedSize;
        uint32                  DependenciesStart;
        uint32                  DependenciesNum;
        uint32
};

//Tier correction.
classFHierarchyFixup
{
public:
        FHierarchyFixup()       {}

        FHierarchyFixup( uint32 InPageIndex, uint32 NodeIndex, uint32 ChildIndex, uint32
InClusterGroupPartStartIndex, uint32 PageDependencyStart, uint32 PageDependencyNum )
        {
                PageIndex = InPageIndex;
                HierarchyNodeAndChildIndex = (NodeIndex << MAX_HIERACHY_CHILDREN_BITS) | ChildIndex;

                ClusterGroupPartStartIndex = InClusterGroupPartStartIndex; PageDependencyStartAndNum =
                (PageDependencyStart << MAX_GROUP_PARTS_BITS) | PageDependencyNum;

        }

          uint32GetPageIndex()const uint32                                  {returnPageIndex; }
          GetNodeIndex()const                                               {returnHierarchyNodeAndChildIndex
 > >MAX_HIERACHY_CHILDREN_BITS; }
         uint32GetChildIndex()const                                        {returnHierarchyNodeAndChildIndex &
 ( MAX_HIERACHY_CHILDREN -1); }
        uint32GetClusterGroupPartStartIndex()const uint32                  {returnClusterGroupPartStartIndex; }
        GetPageDependencyStart()const                                       {returnPageDependencyStartAndNum >>
        MAX_GROUP_PARTS_BITS; }
        uint32GetPageDependencyNum()const &                                 {returnPageDependencyStartAndNum
        MAX_GROUP_PARTS_MASK; }
```

```
        uint32              PageIndex;              uint32
        HierarchyNodeAndChildIndex;         uint32
        ClusterGroupPartStartIndex;         uint32
        PageDependencyStartAndNum;
};

// ClusterCorrection.
classFClusterFixup
{
public:
        FClusterFixup()         {}

        FClusterFixup( uint32 PageIndex, uint32 ClusterIndex, uint32 PageDependencyStart, uint32
        PageDependencyNum )
        {
                PageAndClusterIndex = ( PageIndex << MAX_CLUSTERS_PER_PAGE_BITS ) | ClusterIndex;
                PageDependencyStartAndNum = (PageDependencyStart << MAX_GROUP_PARTS_BITS) |
                PageDependencyNum;
        }

        uint32GetPageIndex()const                               {returnPageAndClusterIndex >>
        MAX_CLUSTERS_PER_PAGE_BITS; }
        uint32GetClusterIndex()const                            {returnPageAndClusterIndex &
(MAX_CLUSTERS_PER_PAGE -1u); }
        uint32GetPageDependencyStart()const                     {returnPageDependencyStartAndNum >>
        MAX_GROUP_PARTS_BITS; }
        uint32GetPageDependencyNum()const                       {returnPageDependencyStartAndNum &
        MAX_GROUP_PARTS_MASK; }

        uint32    PageAndClusterIndex;
        uint32    PageDependencyStartAndNum;
};

//Page header.
structFPageDiskHeader
{
        uint32    GpuSize;
        uint32    NumClusters;
        uint32    NumRawFloat4s;
        uint32    NumTexCoords;
        uint32    DecodeInfoOffset;
        uint32    StripBitmaskOffset;
        uint32    VertexRefBitmaskOffset;
};

// ClusterDisk head.
structFClusterDiskHeader {

        uint32              IndexDataOffset;                uint32
        VertexRefDataOffset;                        uint32
        PositionDataOffset;                         uint32
        AttributeDataOffset;                        uint32
        NumPrevRefVerticesBeforeDwords;             uint32
        NumPrevNewVerticesBeforeDwords;
};

// ChunkCorrection.
classFFixupChunk                //TODO: rename to somethingelse
```

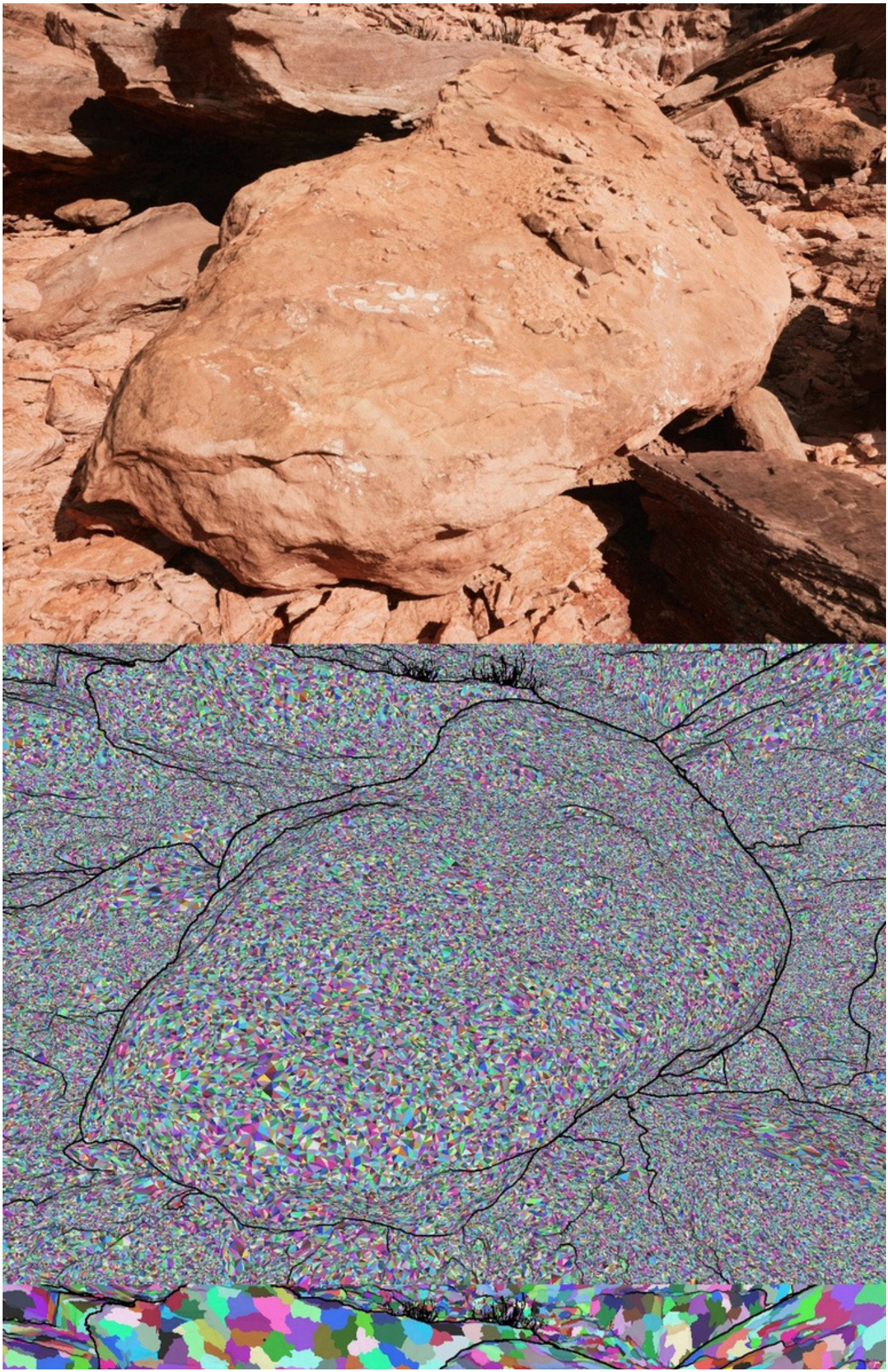
```
{
public:
	struct	FHeader
	{
		uint16 NumClusters =0; uint16
		NumHierachyFixups =0; uint16
		NumClusterFixups =0; uint16 Pad =0; }
		Header;


	uint8 Data[sizeof(FHierarchyFixup) * MAX_CLUSTERS_PER_PAGE +sizeof( FClusterFixup )
* MAX_CLUSTERS_PER_PAGE ];

	FClusterFixup&			GetClusterFixup( uint32 Index )const{ check( Index <
Header.NumClusterFixups );		return( (FClusterFixup*)( Data + Header.NumHierachyFixups *
sizeof( FHierarchyFixup ) ) )[ Index ]; }
	FHierarchyFixup&		GetHierarchyFixup( uint32 Index )const{ check( Index <
Header.NumHierachyFixups );return((FHierarchyFixup*)Data)[ Index ]; }
	uint32				GetSize()const{return sizeof( Header ) +
Header.NumHierachyFixups *sizeof( FHierarchyFixup ) + Header.NumClusterFixups *sizeof( FClusterFixup ); }

};

//Instance drawing parameters.
structFInstanceDraw
{
	uint32	InstanceId;
	uint32	ViewId;
};

// NaniteRendering resources.
structFResources
{
	//Persistent state.
	TArray<uint8	>
RootClusterPage;		// Root page is loaded on
resource load, so we always have something to draw.
	FByteBulkData
StreamableClusterPages;		// Remaining pages are
streamed on	demand.
	TArray< uint16 >
ImposterAtlas;
	TArray<	FPackedHierarchyNode	>	HierarchyNodes;
	TArray<	uint32 >
HierarchyRootOffsets;
	TArray< FPageStreamingState	>	PageStreamingStates;
	TArray<	uint32 >
PageDependencies;
	int32
PositionPrecision =0; =false;
	bool	bLZCompressed

	//Runtime status.
	uint32	RuntimeResourceID	= 0xFFFFFFFFu;
	int32	HierarchyOffset	= INDEX_NONE;
	int32	RootPageIndex	= INDEX_NONE;
	uint32	NumHierarchyNodes	=0;

	(.....)

	ENGINE_API void InitResources();
	ENGINE_API bool ReleaseResources();
	ENGINE_APIvoid Serialize(FArchive& Ar, UObject* Owner);
};
```

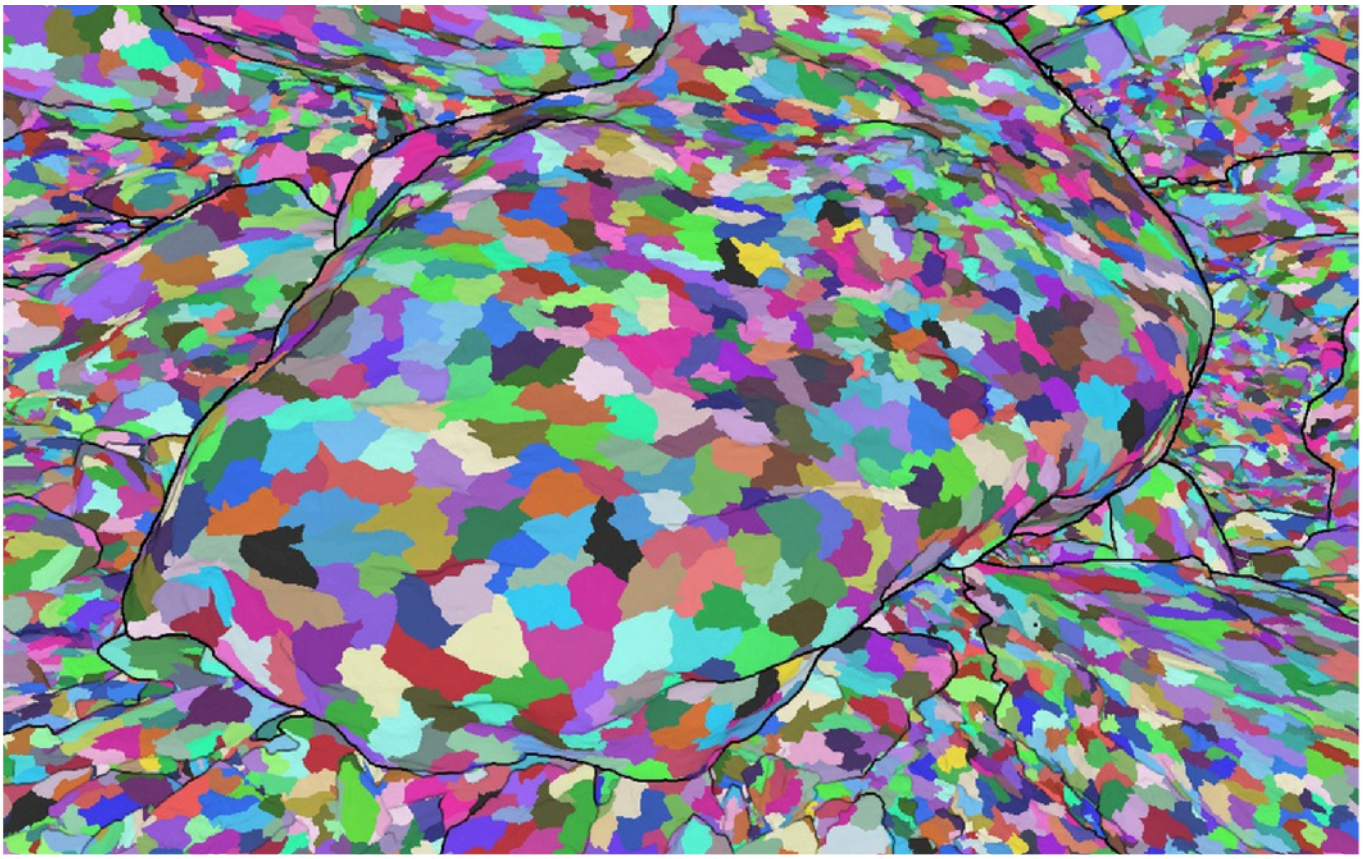
```
N//GanPiUteendBuffer, included Resource data.
classFGlobalResources: public FRenderResource {

public:
	structPassBuffers
	{
		//Candidate (i.e., unpruned) nodes andCluster
		Buffer. TRefCountPtr<FRDGPooledBuffer>	CandidateNodesAndClustersBuffer;
		TRefCountPtr<FRDGPooledBuffer>		StatsRasterizeArgsSWHWBuffer;

	};

	uint32 StatsRenderFlags =0; uint32
	StatsDebugFlags =0;

public:
	virtualvoid InitRHI() override; virtualvoid
	ReleaseRHI() override;

	ENGINE_APIvoid		Update(FRDGBuilder& GraphBuilder);// Called once per frame before
any Nanite rendering has occurred.

	ENGINE_APIstaticuint32GetMaxCandidateClusters();ENGINE_APIstatic
	uint32GetMaxVisibleClusters();ENGINE_APIstaticuint32GetMaxNodes
	();

	(.....)

private:
	PassBuffers	MainPassBuffers;
	PassBuffers	PostPassBuffers;

	classFVertexFactory*VertexFactory= nullptr;

	TRefCountPtr<FRDGPooledBuffer> StatsBuffer;

	// Dummy structured buffer with stride8
	TRefCountPtr<FRDGPooledBuffer> StructureBufferStride8;

#if	NANITE_USE_SCRATCH_BUFFERS TRefCountPtr<FRDGPooledBuffer>
	PrimaryVisibleClustersBuffer; // Used for scratch memory (transient only)
	TRefCountPtr<FRDGPooledBuffer> TRefCountPtr<FRDGPooledBuffer>
							ScratchVisibleClustersBuffer;
							ScratchOccludedInstancesBuffer;
#endif
};

externENGINE_API TGlobalResource<FGlobalResources>GGlobalResources;

}// namespace Nanite
```

6.4.1.4 Cluster, ClusterGroup, Page

Due to the constructionNaniteThere are many concepts involved in data, so I will focus on explaining them here.

NaniteThe most core and basic concept involved is**Cluster**, oneClusteris a set of adjacent triangles:





Top: Normal rendering; Middle: Triangle visualization; Bottom:*Cluster* Visualization.

ClusterCan be adjacentClusteror adjacentLODofClusterDynamic batching makes the picture harmonious and does not produce obvious jumps.this video: Cluster technology is not original to UE, but has been used by Ubisoft and Frostbite Engine earlier. For details, please refer to the papers:GPU-Driven Rendering PipelineandOptimizing the Graphics Pipeline with Compute.

~~The following are the def~~initions of Cluster and other basic types:

```
// Engine\Source\Developer\NaniteBuilder\Private\Cluster.h

//Grid clustering, divides the model into several clusters.

classFCluster
{
public:
        FCluster();
        FCluster( FCluster& SrcCluster, uint32 TriBegin, uint32 TriEnd,constTArray<uint32
> & TriIndexes );
        FCluster(constTArray<constFCluster*, TInlineAllocator<16> >& MergeList ); FCluster(constTArray<
        FStaticMeshBuildVertex >& InVerts,constTArrayView<const uint32 >& InIndexes,

constTArrayView<constint32 >& InMaterialIndexes,constTBitArray<>& InBoundaryEdges,uint32 TriBegin, uint32 TriEnd,
constTArray< uint32 >& TriIndexes, uint32 NumTexCoords,boolbHasColors );


        //simplifyCluster,The desired number of triangles can be specified.
        float Simplify( uint32 NumTris ); //SplitCluster.

        void Split(FGraphPartitioner& Partitioner)const;
```

```
    (......)

    static const uint32 ClusterSize =128;

    //counter.
    uint32 NumVerts = 0;
    uint32 NumTris = 0;
    bool NumTexCoords = 0;
    bool bHasColors = false;

    //NetworTA< float > TArray< uint32 > TArray< int32 > Verts; // vertex
    Indexes;// index
    TBitArray<> MaterialIndexes; //Material index.
    TBitArray<> BoundaryEdges; //Boundary edge.
    uint32 ExternalEdges; //Extended edge.
    NumExternalEdges; //The number of extended edges.

    TMap< uint32, uint32 > AdjacentClusters;//AdjacentCluster.

    //Bounding box data.
    FBounds Bounds; // Bounding box.
    FSphere SphereBounds;
    FSphere LODBounds;
    FVector MeshBoundsMin; //Grid bounding box.
    FVector MeshBoundsDelta;

    float SurfaceArea =0.0f;
    uint32 GUID = 0;
    int32 MipLevel = 0;

    //Data that quantifies position.
    TArray<FIntVector> QuantizedPositions;
    FIntVector QuantizedPosStart = {0u,0u,0u}; =0u;
    uint32 QuantizedPosShift
    FIntVector QuantizedPosBits = {};

    float EdgeLength =0.0f;
    float LODError= 0.0f;

    //LocatedGroupdata.
    uint32 GroupIndex = MAX_uint32;
    uint32 GroupPartIndex = MAX_uint32;
    uint32 GeneratingGroupIndex= MAX_uint32;

    //Material range.
    TArray<FMaterialRange, TInlineAllocator<4>> MaterialRanges;

    //Banded index data.
    FStripDesc StripDesc;
    TArray<uint8> StripIndexData;
};

// Engine\Source\Developer\NaniteBuilder\Private\ClusterDAG.h

//Cluster group, a collection of severalCluster.
struct FClusterGroup
```

```
{
	//	Bounding box .
	FSphere FSphere
	//Error .float
	float
	//Hierarchical and grid indexing.
	int32 uint32

	//Page table index.
	uint32 uint32
	//The subnode index.
	TArray<uint32
	>

	friend FArchive& operator<<(FArchive& Ar, FClusterGroup& Group);
};

// Engine\Source\Developer\NaniteBuilder\Private\NaniteEncode.cpp

//FClusterGroupAll or part of the demerged
structFClusterGroupPart {

	TArray<uint32>	Clusters;	//	Reordering is possible during page allocation, so a list needs to be stored here. Bounding
	FBounds	uint32	Bounds;	//	box.
	uint32	uint32	PageIndex;	//	Page table index.
	uint32	uint32	GroupIndex;	//	LocatedGroupindex.
	HierarchyNodeIndex;	//Hierarchy node index.
	HierarchyChildIndex;	//	Hierarchy subnode index.
	PageClusterOffset;	//	Page ListClusterList offset.
};

//Part of a page table.
structFPageSections
{
	uint32	Cluster	= 0;
	uint32	MaterialTable	= 0;
	uint32	DecodeInfo	= 0;
	uint32	Index	= 0;
	uint32	Position	= 0;
	uint32	Attribute	= 0;

	uint32GetMaterialTableSize()const uint32	{returnAlign(MaterialTable,16); } {return0; }
	GetClusterOffset()const uint32
	GetMaterialTableOffset()const uint32	{returnCluster; }
	GetDecodeInfoOffset()const uint32GetIndexOffset	{returnCluster + GetMaterialTableSize(); } {returnCluster +
	()const DecodeInfo; }	GetMaterialTableSize() +

	uint32GetPositionOffset()const DecodeInfo	{returnCluster + GetMaterialTableSize() +
+ Index; }
	uint32GetAttributeOffset()const DecodeInfo	{returnCluster + GetMaterialTableSize() +
+ Index + Position; }
	uint32GetTotal()const	{returnCluster + GetMaterialTableSize() +
DecodeInfo + Index + Position + Attribute; }

	FPageSectionsGetOffsets()const
```

```
	{
		returnFPageSections{ GetClusterOffset(), GetMaterialTableOffset(), GetDecodeInfoOffset(),
GetIndexOffset(), GetPositionOffset(), GetAttributeOffset() };
	}

	voidoperator+=(constFPageSections& Other) {

		Cluster		+ =	Other.Cluster;
		MaterialTable	+ =	Other.MaterialTable;
		DecodeInfo	+ =	Other.DecodeInfo;
		Index		+ =	Other.Index;
		Position	+ =	Other.Position;
		Attribute	+ =	Other.Attribute;
	}
} ;

//ClsuterPage table.
structFPage
{
	uint32	PartsStartIndex =0;//FClusterGroupPartStarting index.
	uint32	PartsNum =0;//FClusterGroupPartquantity.
	uint32	NumClusters =0;// Clusterquantity.

	FPageSections	GpuSizes;//GPUSize.
};

//Encoded information.
structFEncodingInfo
{
	uint32 BitsPerIndex;//The number of bits per index.
	uint32 BitsPerAttribute;//The number of bits per attribute.
	uint32 UVPrec;//UVAccuracy.

	uint32		ColorMode;//Color mode.
	FIntVector4	ColorMin;//Minimum color.
	FIntVector4 ColorBits;//Number of color digits.

	FPageSections GpuSizes;//GPUSize.

	//UVEncoded information.
	FGeometryEncodingUVInfo UVInfos[MAX_NANITE_UVS];
};

// Cluster HierarchyThe intermediate node is used to build
Hierarchy. structFIntermediateNode {

	uint32		PartIndex	= MAX_uint32;//FClusterGroupPartindex. =
MAX_int32;// MipTier.
	uint32		MipLevel
	bool		bLeaf	=false;//Whether it is a leaf node. //
Bounding box.

	FBounds		Bound;
	TArray< uint32 >	Children;//	List of child nodes.
};

// Engine\Source\Developer\NaniteBuilder\Private\ImposterAtlas.h

// ClusterRasterized atlas.
classFImposterAtlas
```

```
{
public:
	static	constexpr	uint32	AtlasSize		= 12;
	static	constexpr	uint32	TileSize		= 12;

			FImposterAtlas( TArray< uint16 >& InPixels,constFBounds& MeshBounds );
	//Rasterization SpecificationClusterAll triangles ofFImposterAtlas.
	void		Rasterize(constFIntPoint& TilePos,constFCluster&Cluster, uint32
ClusterIndex	) ;

private:
	TArray<uint16>&		Pixels;

	FVector		BoundsCenter;
	FVector		BoundsExtent;

	FMatrix		GetLocalToImposter(constFIntPoint& TilePos )const;
};
```

6.4.2 Nanite Data Construction

This section mainly describes the preprocessing performed by Nanite before rendering, including the construction and calling process of Nanite static data.

6.4.2.1 BuildNaniteFromHiResSourceModel

Nanite builds the required data from the highest resolution model through the BuildNaniteFromHiResSourceModel interface, which is similar to the FStaticMeshBuilder::Build() interface, but ignores the face reduction process, which is called Nanite-fractional-cut. The specific process is as follows :

```
// Engine\Source\Developer\MeshBuilder\Private\StaticMeshBuilder.cpp

static bool BuildNaniteFromHiResSourceModel(
	UStaticMesh* StaticMesh,
	constFMeshNaniteSettings NaniteSettings,
	FBoxSphereBounds& HiResBoundsOut,
	Nanite::FResources& NaniteResourcesOut)
{
	//Ignore static meshes that do not have high resolution.
	if(ensure(StaticMesh->IsHiResMeshDescriptionValid()) ==false) {

		returnfalse;
	}

	TRACE_CPUPROFILER_EVENT_SCOPE(FStaticMeshBuilder::BuildNaniteFromHiResSourceModel);

	//Get model data

	FMeshDescription HiResMeshDescription = *StaticMesh->GetHiResMeshDescription();
	FStaticMeshSourceModel& HiResSrcModel = StaticMesh->GetHiResSourceModel(); FMeshBuildSettings&
	HiResBuildSettings = HiResSrcModel.BuildSettings;

	//Calculate tangents, light maps, etc.UwaVit	.
	FMeshDescriptionHelperMeshDescriptionHelper(&HiResBuildSettings);
```

```
MeshDescriptionHelper.SetupRenderMeshDescription(StaticMesh, HiResMeshDescription);

//Build a temporaryRenderDatadata to be passed to subsequentNaniteBuild phase.
FStaticMeshRenderData HiResTempRenderData;
HiResTempRenderData.AllocateLODResources(1); //Note that the
index obtained is0ofLODdata (i.e. the highest resolution data).

FStaticMeshLODResources& HiResStaticMeshLOD = HiResTempRenderData.LODResources[0];
HiResStaticMeshLOD.MaxDeviation =0.0f;
//PreparePerSectionIndicesArray, to optimize theGPUThe index buffer.


TArray<TArray<uint32>> PerSectionIndices;
PerSectionIndices.AddDefaulted(HiResMeshDescription.PolygonGroups().Num());
HiResStaticMeshLOD.Sections.Empty(HiResMeshDescription.PolygonGroups().Num());

                                        WedgeMaporRemapVerts
//    Constructing vertex and index buffers does. not require
TArray<int32>       WedgeMap, RemapVerts;
TArray<FStaticMeshBuildVertex>          StaticMeshBuildVertices;
BuildVertexBuffer(StaticMesh, HiResMeshDescription, HiResBuildSettings, WedgeMap,
HiResStaticMeshLOD.Sections, PerSectionIndices,                 StaticMeshBuildVertices,
MeshDescriptionHelper.GetOverlappingCorners(),                  RemapVerts);
WedgeMap.Empty();

constuint32 NumTextureCoord =
HiResMeshDescription.VertexInstanceAttributes().GetAttributesRef<FVector2D>
(MeshAttribute::VertexInstance::TextureCoordinate).GetNumChannels();

//Only rendering data and vertex data need to be used, so they can be cleaned up.

MeshDescription. HiResMeshDescription.Empty();

//Link bysectionThe index buffer.

TArray<uint32> CombinedIndices; boolbNeeds32BitIndices =
false; BuildCombinedSectionIndices(PerSectionIndices,
bNeeds32BitIndices);                                    HiResStaticMeshLOD,     CombinedIndices,


//existNaniteCompute bounding boxes from the high-resolution mesh before building, as it will modify
                                                        StaticMeshBuildVertices.
ComputeBoundsFromVertexList(StaticMeshBuildVertices, HiResBoundsOut.Origin,
HiResBoundsOut.BoxExtent, HiResBoundsOut.SphereRadius);

// NaniteBuild requirementssectionThe material index has been changed fromSectionInfoMapparse it out because the index is baked into
FMaterialTriangles.
for(int32 SectionIndex =0; SectionIndex < HiResStaticMeshLOD.Sections.Num(); SectionIndex++)

{
    HiResStaticMeshLOD.Sections[SectionIndex].MaterialIndex = StaticMesh-
> GetSectionInfoMap().Get(0, SectionIndex).MaterialIndex; }


//RunNaniteBuild.
{

TRACE_CPUPROFILER_EVENT_SCOPE(FStaticMeshBuilder::BuildNaniteFromHiResSourceModel::Nanite);

    Nanite::IBuilderModule& NaniteBuilderModule = Nanite::IBuilderModule::Get(); if(!
    NaniteBuilderModule.Build(NaniteResourcesOut, StaticMeshBuildVertices, CombinedIndices,
HiResStaticMeshLOD.Sections, NumTextureCoord, NaniteSettings))
    {
        UE_LOG(LogStaticMesh, Error, TEXT("Failed to build Nanite for HiRes static
```

```
mesh. See previous line(s) for details."));
                returnfalse;
            }
        }

        returntrue;
}
```

The above code involves several important interfaces, which are analyzed below:

```
// Engine\Source\Runtime\Engine\Private\StaticMesh.cpp

//Is there a valid high-resolution grid?
bool UStaticMesh::IsHiResMeshDescriptionValid()const {

        constFStaticMeshSourceModel& SourceModel = GetHiResSourceModel(); return
        SourceModel.IsMeshDescriptionValid();
}


// Engine\Source\Developer\MeshBuilder\Private\MeshDescriptionHelper.cpp

void FMeshDescriptionHelper::SetupRenderMeshDescription(UObject* Owner, FMeshDescription&
RenderMeshDescription)
{
        TRACE_CPUPROFILER_EVENT_SCOPE(FMeshDescriptionHelper::GetRenderMeshDescription);

        UStaticMesh* StaticMesh = Cast<UStaticMesh>(Owner);

        const boolbNaniteBuildEnabled = StaticMesh->NaniteSettings.bEnabled; float
        ComparisonThreshold = (BuildSettings->bRemoveDegenerates && !bNaniteBuildEnabled) ?
THRESH_POINTS_ARE_SAME :0.0f;

        //Ensures polygon normals, tangents, and binormals are calculated, and also fromRender mesh descriptionDeleted degenerate
        three-piece set. FStaticMeshOperations::ComputeTriangleTangentsAndNormals(RenderMeshDescription,
ComparisonThreshold);

        FVertexInstanceArray& VertexInstanceArray = RenderMeshDescription.VertexInstances();

        FStaticMeshAttributesAttributes(RenderMeshDescription); TVertexInstanceAttributesRef<FVector> Normals =
        Attributes.GetVertexInstanceNormals(); TVertexInstanceAttributesRef<FVector>
                                                    Tangents =
Attributes.GetVertexInstanceTangents();
        TVertexInstanceAttributesRef<float>          BinormalSigns =
        Attributes.GetVertexInstanceBinormalSigns();

        //Find overlapping vertices and speed up adjacency.
        FStaticMeshOperations::FindOverlappingCorners(OverlappingCorners,
RenderMeshDescription, ComparisonThreshold);

        //Static meshes always blend normals of overlapping corners.
        EComputeNTBsFlags ComputeNTBsOptions = EComputeNTBsFlags::BlendOverlappingNormals;
        ComputeNTBsOptions |= BuildSettings->bComputeWeightedNormals?
EComputeNTBsFlags::WeightedNTBs : EComputeNTBsFlags::None;
        ComputeNTBsOptions |= BuildSettings->bRecomputeNormals ? EComputeNTBsFlags::Normals :
        EComputeNTBsFlags::None;
        ComputeNTBsOptions |= BuildSettings->bUseMikkTSpace ? EComputeNTBsFlags::UseMikkTSpace
```

```
: EComputeNTBsFlags::None;

    //  NaniteThe grid does not calculate tangent data.
    if  (!bNaniteBuildEnabled)
    {
        ComputeNTBsOptions |= BuildSettings->bRemoveDegenerates ?
EComputeNTBsFlags::IgnoreDegenerateTriangles : EComputeNTBsFlags::None;
        ComputeNTBsOptions |= BuildSettings->bRecomputeTangents ?
EComputeNTBsFlags::Tangents : EComputeNTBsFlags::None;
    }

    //Compute any missing normals or tangents.
    FStaticMeshOperations::ComputeTangentsAndNormals(RenderMeshDescription,
    ComputeNTBsOptions);

    //Generating a light mapUV.
    if(BuildSettings->bGenerateLightmapUVs && VertexInstanceArray.Num() >0) {

        TVertexInstanceAttributesRef<FVector2D> VertexInstanceUVs =
        Attributes.GetVertexInstanceUVs();
        int32 NumIndices = VertexInstanceUVs.GetNumChannels(); //Verify the
        src light map channel
        if(BuildSettings->SrcLightmapIndex >= NumIndices) {

            BuildSettings->SrcLightmapIndex =0;
        }
        //Verify the destination light map cha n n
        if ( B uild Settings -> D st L ightmap Index { > = N

            //Make sure we do not add illegal UV Channel index
            if(BuildSettings->DstLightmapIndex >= MAX_MESH_TEXTURE_COORDS_MD) {

                BuildSettings->DstLightmapIndex = MAX_MESH_TEXTURE_COORDS_MD -1;
            }

            //Add some unused UVChannel to the mesh description for the lightmapUVs
            VertexInstanceUVs.SetNumChannels(BuildSettings->DstLightmapIndex +1); BuildSettings-
            >DstLightmapIndex = NumIndices;
        }
        FS tatic M esh O perations : : C reate L ight M ap U V L
            BuildSettings->SrcLightmapIndex, BuildSettings-
            > DstLightmapIndex, BuildSettings->MinLightmapResolution,
            (ELightmapUVVersion)StaticMesh->GetLightmapUVVersion(),
            OverlappingCorners);

    }
}


// Engine\Source\Developer\MeshBuilder\Private\StaticMeshBuilder.cpp

//Build the vertex buffer.
void BuildVertexBuffer(
      UStaticMesh  * StaticMesh
    , const FMeshDescription&  MeshDescription
    , const FMeshBuildSettings  BuildSettings
    ,  TArray<int32>&  OutWedgeMap  ,
    FStaticMeshSectionArray&  OutSections
```

```
        , TArray<TArray<uint32> >& OutPerSectionIndices
        , TArray< FStaticMeshBuildVertex >& StaticMeshBuildVertices
        , constFOverlappingCorners&            OverlappingCorners
        , TArray<int32>& RemapVerts)
{
        TRACE_CPUPROFILER_EVENT_SCOPE(BuildVertexBuffer);

        TArray<int32> RemapVertexInstanceID;
        //Sets the vertex buffer element.
        constint32 NumVertexInstances = MeshDescription.VertexInstances().GetArraySize();
        StaticMeshBuildVertices.Reserve(NumVertexInstances);


        FStaticMeshConstAttributesAttributes(MeshDescription);

        TPolygonGroupAttributesConstRef<FName> PolygonGroupImportedMaterialSlotNames =
        Attributes.GetPolygonGroupMaterialSlotNames();
        TVertexAttributesConstRef<FVector> VertexPositions = Attributes.GetVertexPositions();
        TVertexInstanceAttributesConstRef<FVector>                VertexInstanceNormals        =
Attributes.GetVertexInstanceNormals();
        TVertexInstanceAttributesConstRef<FVector>                VertexInstanceTangents         =
        Attributes.GetVertexInstanceTangents();
        TVertexInstanceAttributesConstRef<float> VertexInstanceBinormalSigns                        =
        Attributes.GetVertexInstanceBinormalSigns();
        TVertexInstanceAttributesConstRef<FVector4>                VertexInstanceColors =
        Attributes.GetVertexInstanceColors();
        TVertexInstanceAttributesConstRef<FVector2D>                VertexInstanceUVs =
        Attributes.GetVertexInstanceUVs();


        const boolbHasColors = VertexInstanceColors.IsValid(); const boolbIgnoreTangents =
        StaticMesh->NaniteSettings.bEnabled;

        constuint32 NumTextureCoord = VertexInstanceUVs.GetNumChannels(); constFMatrix
        ScaleMatrix = FScaleMatrix(BuildSettings.BuildScale3D).Inverse().GetTransposed();


        TMap<FPolygonGroupID, int32> PolygonGroupToSectionIndex;

           for(constFPolygonGroupID PolygonGroupID :
           MeshDescription.PolygonGroups().GetElementIDs())
        {
              int32& SectionIndex = PolygonGroupToSectionIndex.FindOrAdd(PolygonGroupID); SectionIndex =
              OutSections.Add(FStaticMeshSection());
              FStaticMeshSection& StaticMeshSection = OutSections[SectionIndex];
              StaticMeshSection.MaterialIndex = StaticMesh-
> GetMaterialIndexFromImportedMaterialSlotName(PolygonGroupImportedMaterialSlotNames[Polygo nGroupID]);

              if(StaticMeshSection.MaterialIndex == INDEX_NONE) {

                    StaticMeshSection.MaterialIndex = PolygonGroupID.GetValue();
              }
        }

        int32 ReserveIndicesCount = MeshDescription.Triangles().Num() *3;

        //Fill the remap array.
        RemapVerts.AddZeroed(ReserveIndicesCount); for
        (int32& RemapIndex : RemapVerts) {
```

```
            RemapIndex = INDEX_NONE;
        }

        //Initialize wedge tableOutWedgeMap
        OutWedgeMap.Reset();
        OutWedgeMap.AddZeroed(ReserveIndicesCount);

        floatVertexComparisonThreshold = BuildSettings.bRemoveDegenerates ?
THRESH_POINTS_ARE_SAME :0.0f;

        int32 WedgeIndex =0;
        for(constFTriangleID TriangleID : MeshDescription.Triangles().GetElementIDs()) {

            constFPolygonGroupID PolygonGroupID =
            MeshDescription.GetTrianglePolygonGroup(TriangleID);
            constint32 SectionIndex = PolygonGroupToSectionIndex[PolygonGroupID]; TArray<uint32>&
            SectionIndices = OutPerSectionIndices[SectionIndex];

                TArrayView<constFVertexID> VertexIDs =
                MeshDescription.GetTriangleVertices(TriangleID);

            FVector CornerPositions[3];
            for(int32 TriVert =0; TriVert <3; ++TriVert) {

                CornerPositions[TriVert] = VertexPositions[VertexIDs[TriVert]];
            }
            FO verlapping T hresholds                                                    Overlapping Ther                                                    s  h
            O verlapping T hresholds . T hreshold P ositio //Do not process merged triangles.                                                    n

            if(PointsEqual(CornerPositions[0], CornerPositions[1], OverlappingThresholds)
                || PointsEqual(CornerPositions[0], CornerPositions[2], OverlappingThresholds) ||
                PointsEqual(CornerPositions[1], CornerPositions[2], OverlappingThresholds))
            {
                WedgeIndex +=3;
                continue;
            }

            TArrayView<constFVertexInstanceID> VertexInstanceIDs =
            MeshDescription.GetTriangleVertexInstances(TriangleID);
            for(int32 TriVert =0; TriVert <3; ++TriVert, ++WedgeIndex) {

                constFVertexInstanceID VertexInstanceID = VertexInstanceIDs[TriVert]; constFVector&
                VertexPosition = CornerPositions[TriVert];
                constFVector& VertexInstanceNormal = VertexInstanceNormals[VertexInstanceID]; constFVector&
                VertexInstanceTangent =
VertexInstanceTangents[VertexInstanceID];
                const floatVertexInstanceBinormalSign =
VertexInstanceBinormalSigns[VertexInstanceID];

                FStaticMeshBuildVertex StaticMeshVertex;

                StaticMeshVertex.Position = VertexPosition * BuildSettings.BuildScale3D;
                //in the case ofNaniteGrid, directly assign fixed tangents and
                bitangents. if( bIgnoreTangents ) {

                    StaticMeshVertex.TangentX = FVector(1.0f,0.0f,0.0f);
                    StaticMeshVertex.TangentY = FVector(0.0f,1.0f,0.0f);
                }
```

```
                else
                {
                        StaticMeshVertex.TangentX           =
ScaleMatrix.TransformVector(VertexInstanceTangent).GetSafeNormal();
                        StaticMeshVertex.TangentY =
ScaleMatrix.TransformVector(FVector::CrossProduct(VertexInstanceNormal,
VertexInstanceTangent) * VertexInstanceBinormalSign).GetSafeNormal();
                }
                S tatic M esh V ertex . T angent Z                                                  =
ScaleMatrix.TransformVector(VertexInstanceNormal).GetSafeNormal();

                if(bHasColors)
                {
                        constFVector4& VertexInstanceColor =
VertexInstanceColors[VertexInstanceID];
                        constFLinearColorLinearColor(VertexInstanceColor);
                        StaticMeshVertex.Color          = LinearColor.ToFColor(true);
                }
                else
                {
                        StaticMeshVertex.Color          =  FColor::White;
                }

                constuint32 MaxNumTexCoords = FMath::Min<int32>(MAX_MESH_TEXTURE_COORDS_MD,
MAX_STATIC_TEXCOORDS);
                for(uint32 UVIndex =0; UVIndex < MaxNumTexCoords; ++UVIndex) {

                        if(UVIndex < NumTextureCoord) {

                                StaticMeshVertex.UVs[UVIndex] =
VertexInstanceUVs.Get(VertexInstanceID, UVIndex);
                        }
                        else
                        {
                                StaticMeshVertex.UVs[UVIndex] = FVector2D(0.0f,0.0f);
                        }
                }
                //No duplicate vertex instances are created. Use the already
                constructedWedgeIndex constTArray<int32>& DupVerts =

OverlappingCorners.FindIfOverlapping(WedgeIndex);

                int32 Index = INDEX_NONE;
                for(int32 k =0; k < DupVerts.Num(); k++) {

                        if(DupVerts[k] >= WedgeIndex) {

                                break;
                        }
                        int 3 2                 Location                        =       R emap Verts . I s V ali                                        d I n
RemapVerts[DupVerts[k]] : INDEX_NONE;
                        if(Location != INDEX_NONE && AreVerticesEqual(StaticMeshVertex,
StaticMeshBuildVertices[Location], VertexComparisonThreshold))
                        {
                                Index = Location;
                                break;
                        }
                }
```

```
                if(Index == INDEX_NONE) {

                        Index = StaticMeshBuildVertices.Add(StaticMeshVertex);
                }
                R emap V erts [ W edge I ndex ] O ut W edge M ap [ W edge I ndex ] S ecti=dnmdex ;
                ndices . A dd ( I ndex                                                =      I ndex ;
                                                                                             ) ;
        }
    }

    //Optimize before setting the buffer.

    if(NumVertexInstances <100000*3) {


BuildOptimizationHelper::CacheOptimizeVertexAndIndexBuffer(StaticMeshBuildVertices, OutPerSectionIndices,
OutWedgeMap);
    }
}

//Build a combinationSectionindex.

static void BuildCombinedSectionIndices(
    constTArray<TArray<uint32>>&    PerSectionIndices,
    FStaticMeshLODResources&    StaticMeshLODInOut,
    TArray<uint32>&     CombinedIndicesOut,     bool&
    bNeeds32BitIndicesOut )
{
    bNeeds32BitIndicesOut =false;
    for(int32 SectionIndex =0; SectionIndex < StaticMeshLODInOut.Sections.Num(); SectionIndex++)

    {
        FStaticMeshSection& Section = StaticMeshLODInOut.Sections[SectionIndex]; constTArray<uint32>&
        SectionIndices = PerSectionIndices[SectionIndex]; Section.FirstIndex =0; Section.NumTriangles =0;
        Section.MinVertexIndex =0; Section.MaxVertexIndex =0;




        if(SectionIndices.Num()) {

            Section.FirstIndex = CombinedIndicesOut.Num();
            Section.NumTriangles = SectionIndices.Num() /3;

            CombinedIndicesOut.AddUninitialized(SectionIndices.Num()); uint32* DestPtr =
            &CombinedIndicesOut[Section.FirstIndex]; uint32const* SrcPtr =
            SectionIndices.GetData();

            Section.MinVertexIndex          =   * SrcPtr;
            Section.MaxVertexIndex          =   * SrcPtr;

            for(int32 Index =0; Index < SectionIndices.Num(); Index++) {

                uint32 VertIndex = *SrcPtr++;

                bNeeds32BitIndicesOut |= (VertIndex > MAX_uint16);
                Section.MinVertexIndex          = FMath::Min<uint32>(VertIndex,
Section.MinVertexIndex);
                Section.MaxVertexIndex          = FMath::Max<uint32>(VertIndex,
Section.MaxVertexIndex);
```

```
                        * DestPtr++ = VertIndex;
                    }
                }
            }
        }
    }

    //Calculate bounding box and sphere based on vertices
    static void ComputeBoundsFromVertexList(constTArray<FStaticMeshBuildVertex>& Vertices, FVector& OriginOut,
    FVector& ExtentOut,float& RadiusOut) {

        //Calculate bounding box
        FBoxBoundingBox(ForceInit);
        for(int32 VertexIndex =0; VertexIndex < Vertices.Num(); VertexIndex++) {


            BoundingBox += Vertices[VertexIndex].Position;
        }
        B ounding B ox . G et C enter A nd Extents ( O rigin O ut ,

        //Calculate the sphere and use the center of the bounding box as the center of the sphere.
        RadiusOut =0.0f;
        for(int32 VertexIndex =0; VertexIndex < Vertices.Num(); VertexIndex++) {


            RadiusOut = FMath::Max((Vertices[VertexIndex].Position-OriginOut).Size(), RadiusOut);

        }
    }
```

Much of the above logic is similar to that of a normal static mesh, but there are a few differences:

- Nanite's source model comes from the ultra-high resolution model HiResSourceModel. Nanite
- meshes ignore the calculation of tangents, bitangents, and face reduction. Finally,
- Nanite::IBuilderModule::Build will be called to actually build the Nanite grid data. See the next section for details.

6.4.2.2 BuildNaniteData

This section will explain the construction process of the Nanite grid.

```
// Engine\Source\Developer\NaniteBuilder\Private\NaniteBuilder.cpp

bool FBuilderModule::Build(
        FResources& Resources,
        TArray< FStaticMeshBuildVertex>& TArray< Vertices,
        uint32 >& TriangleIndices,
        TArray< FStaticMeshSection, TInlineAllocator<1>>& Sections, uint32
                NumTexCoords,
        const    FMeshNaniteSettings& Settings)
{
        TRACE_CPUPROFILER_EVENT_SCOPE(Nanite::Build);

        check(Sections.Num() >0&& Sections.Num() <=64);

        //Build an associative array of triangle indices and material indices.
        TArray<int32> MaterialIndices; {


                TRACE_CPUPROFILER_EVENT_SCOPE(Nanite::BuildSections);
```

```
            //The number of material indices is the same as the number of triangles.
            MaterialIndices.Reserve(TriangleIndices.Num() /3);
            for(int32 SectionIndex =0; SectionIndex < Sections.Num(); SectionIndex++) {

                FStaticMeshSection& Section = Sections[SectionIndex];

                check(Section.MaterialIndex != INDEX_NONE); for(uint32 i =0; i <
                Section.NumTriangles; ++i) {

                    MaterialIndices.Add(Section.MaterialIndex);
                }
            }
        }

        TArray<uint32> MeshTriangleCounts;
        MeshTriangleCounts.Add(TriangleIndices.Num() /3);

        //Make sure each triangle has a material index.
        check(MaterialIndices.Num() *3== TriangleIndices.Num());

        //BuildNanitedata.
        returnBuildNaniteData(
            Resources,    Vertices,
            TriangleIndices,
            MaterialIndices,
            MeshTriangleCounts,
            Sections,
            NumTexCoords,
            Settings

        ) ;
}

//BuildNanitedata.
static bool BuildNaniteData(
        FResources& Resources,
        TArray< FStaticMeshBuildVertex >& Verts,//TODO:Do not require this vertex type for all users of Nanite

        TArray<uint32>& Indexes, TArray<int32>&
        MaterialIndexes, TArray<uint32>&
        MeshTriangleCounts,
        TArray< FStaticMeshSection, TInlineAllocator<1> >& Sections, uint32
                NumTexCoords,
        const    FMeshNaniteSettings & Settings
)
{
        TRACE_CPUPROFILER_EVENT_SCOPE(Nanite::BuildData);

        if(NumTexCoords > MAX_NANITE_UVS) NumTexCoords = MAX_NANITE_UVS;

        FBounds          VertexBounds;
        uint32 Channel =255;//Used to check whether there is valid vertex data.

        for(auto& Vert : Verts ) {

            VertexBounds += Vert.Position;

            Channel     &=  Vert.Color.R;
            Channel     &=  Vert.Color.G;
```

```
            Channel    &=  Vert.Color.B;
            Channel    &= Vert.Color.A;
        }

        constuint32 NumMeshes = MeshTriangleCounts.Num();

        //It only has color data when it is not completely white.
        boolbHasColors = Channel !=255;


        TArray< uint32 > ClusterCountPerMesh; TArray<
        FCluster > Clusters;
        {
            uint32 BaseTriangle =0;
            //Traverse allSection,For eachSectionBuild one or moreCluster. for
            (uint32 NumTriangles : MeshTriangleCounts) {

                uint32 NumClustersBefore = Clusters.Num(); if
                (NumTriangles)
                {
                    //    For eachSectionBuild1or moreCluster.UsedTArrayViewConstruct an array of reused data. Analysis will
                    //    be given later.ClusterTrianglesspecific process.
                    ClusterTriangles(Verts, TArrayView<constuint32 >( &Indexes[BaseTriangle
*  3], NumTriangles *3),
                                                        TArrayView<constint32 >(
&MaterialIndexes[BaseTriangle], NumTriangles ),
                                                        Clusters, VertexBounds, NumTexCoords, bHasColors);
                }
                //          Record eachSectionofClusterquantity .
                C luster C ount P er M esh . A dd ( C lusters . N um B ase T riangle + = N um T riangles ;                                    ( )

            }
        }

        constint32 OldTriangleCount = Indexes.Num() /3; constint32
        MinTriCount =2000;
        //Use rough representation (coarse representationReplace the original static grid data.
        const boolbUseCoarseRepresentation = Settings.PercentTriangles <1.0f&& OldTriangleCount >
        MinTriCount;


        //    If you don't use rough representation (coarse     representation)Replaces the original vertex buffer, removing the old copied data.
                                                                In the longerDAGExecuted before the reduce phase to reduce peak memory duration.
        //    Copy it tocluster representation When building multi,ple hugeNanite
        //    This is especially important when using a grid.
        if    (bUseCoarseRepresentation)
        {
            check(MeshTriangleCounts.Num() ==1
            );Verts.Empty();
            Indexes.Empty();
            MaterialIndexes.Empty();
        }

        uint32 Time0 = FPlatformTime::Cycles();

        FBounds MeshBounds;
        TArray<FClusterGroup> Groups;// ClusterList of groups. {

            TRACE_CPUPROFILER_EVENT_SCOPE(Nanite::Build::DAG.Reduce);

            uint32 ClusterStart =0;
```

```
            for(uint32 MeshIndex =0; MeshIndex < NumMeshes; MeshIndex++) {

                uint32 NumClusters = ClusterCountPerMesh[MeshIndex];
                //BuildDAG （Directed Acyclic Graph, directed acyclic graph), by reducing the face and module, and addingClusterandGroup
to the corresponding array.
                BuildDAG(Groups, Clusters, ClusterStart, NumClusters, MeshIndex, MeshBounds
) ;
                ClusterStart += NumClusters;
            }
        }

        uint32 ReduceTime = FPlatformTime::Cycles(); UE_LOG(LogStaticMesh, Log,
        TEXT("Reduce [%.2fs]"), FPlatformTime::ToMilliseconds(ReduceTime - Time0) /
1000.0f);

        //  Use a rough representation.
        if  (bUseCoarseRepresentation)
        {
            constuint32 CoarseStartTime = FPlatformTime::Cycles();
            int32 CoarseTriCount = FMath::Max(MinTriCount, int32((float(OldTriangleCount) *
            Settings.PercentTriangles)));

            TArray<FStaticMeshSection, TInlineAllocator<1>> CoarseSections = Sections;
            //Construct a rough representation.
            BuildCoarseRepresentation(Groups, Clusters, Verts, Indexes, CoarseSections, NumTexCoords,
            CoarseTriCount);


            //  Correct the grid using the coarse grid rangesectioninformation, while respecting the original sequence and preserving the texture. It
            //  does not end up in any specific triangle (due to the extraction process).

            for(FStaticMeshSection& Section : Sections) {

                //For eachsection, trying to find a matching entry in the rough version.
                constFStaticMeshSection* CoarseSection = CoarseSections.FindByPredicate(
                    [&Section](constFStaticMeshSection& CoarseSectionIter)
                {
                    returnCoarseSectionIter.MaterialIndex == Section.MaterialIndex;
                });

                //Find matching entries
                if(CoarseSection != nullptr) {

                    Section.FirstIndex      =  CoarseSection->FirstIndex;
                    Section.NumTriangles    =  CoarseSection->NumTriangles;
                    Section.MinVertexIndex   =   CoarseSection->MinVertexIndex;
                    Section.MaxVertexIndex  =  CoarseSection->MaxVertexIndex;
                }
                //No matching entries were found.
                else
                {
                    //Set a placeholder entry for the part that is removed due to extraction

                    Section.FirstIndex      =  0;
                    Section.NumTriangles    =  0;
                     Section.MinVertexIndex  =  0;
                     Section.MaxVertexIndex  =  0;
                }
            }
```

```
            constuint32 CoarseEndTime = FPlatformTime::Cycles();
            UE_LOG(LogStaticMesh, Log, TEXT("Coarse [%.2fs], original tris: %d, coarse tris: %d"),
FPlatformTime::ToMilliseconds(CoarseEndTime - CoarseStartTime) /1000.0f, OldTriangleCount, CoarseTriCount);

        }

        uint32 EncodeTime0 = FPlatformTime::Cycles();

        //codingNaniteGrid.
        Encode( Resources, Settings, Clusters, Groups, MeshBounds, NumMeshes, NumTexCoords, bHasColors );


        uint32 EncodeTime1 = FPlatformTime::Cycles();
        UE_LOG( LogStaticMesh, Log, TEXT("Encode [%.2fs]"), FPlatformTime::ToMilliseconds( EncodeTime1 -
EncodeTime0 ) /1000.0f);

        //Only generated when there is one gridImposter.
        const boolbGenerateImposter = (NumMeshes ==1); if
        (bGenerateImposter)
        {
            uint32 ImposterStartTime = FPlatformTime::Cycles(); auto&
            RootChildren = Groups.Last().Children;
            // ResourcesofImposterAtlas.
            FImposterAtlasImposterAtlas( Resources.ImposterAtlas, MeshBounds );


            //Parallel generationImposter.
            ParallelFor(FMath::Square(FImposterAtlas::AtlasSize),
                [&](int32       TileIndex)
            {
                FIntPoint       TilePos(
                    TileIndex       %FImposterAtlas::AtlasSize,
                    TileIndex       / FImposterAtlas::AtlasSize);

                //Traverse all subCluster,Rasterize toImposterAtlas.
                for(int32 ClusterIndex =0; ClusterIndex < RootChildren.Num();
ClusterIndex++)
                {
                    ImposterAtlas.Rasterize(TilePos, Clusters[RootChildren[ClusterIndex]],
ClusterIndex);
                }
            });

            UE_LOG(LogStaticMesh, Log, TEXT("Imposter [%.2fs]"),
FPlatformTime::ToMilliseconds(FPlatformTime::Cycles() - ImposterStartTime ) /1000.0f);
        }

        uint32 Time1 = FPlatformTime::Cycles();

        UE_LOG( LogStaticMesh, Log, TEXT("Nanite build [%.2fs]\n"),
FPlatformTime::ToMilliseconds( Time1 - Time0 ) /1000.0f);

        returntrue;
}
```

6.4.2.3 ClusterTriangles

```
//For eachSectionBuild1or moreCluster.
static void ClusterTriangles(
	constTArray< FStaticMeshBuildVertex >& Verts, constTArrayView<
	constuint32 >& Indexes, constTArrayView<constint32 >&
	MaterialIndexes, TArray< FCluster >& Clusters,
                                                // Append
	const	FBounds & MeshBounds,
	uint32 NumTexCoords,
	boolbHasColors		)
{
	uint32 Time0 = FPlatformTime::Cycles();

	LOG_CRC(	Verts );
	LOG_CRC(	Indexes );

	uint32 NumTriangles = Indexes.Num() /3;

	//Shared Edges

	TArray< uint32 > SharedEdges;
	SharedEdges.AddUninitialized(		Indexes.Num()	) ;

	//Boundary Edge

	TBitArray<> BoundaryEdges; BoundaryEdges.Init(false,
	Indexes.Num() );

	//Edge Hashing

	FHashTableEdgeHash(1<< FMath::FloorLog2( Indexes.Num() ), Indexes.Num() );

	//Process edge hashing in parallel.

	ParallelFor( Indexes.Num(),
		[&]( int32 EdgeIndex ) {


			uint32 VertIndex0 = Indexes[ EdgeIndex ]; uint32 VertIndex1 =
			Indexes[ Cycle3( EdgeIndex ) ];

			constFVector& Position0 = Verts[VertIndex0].Position; constFVector&
			Position1 = Verts[VertIndex1].Position;

			uint32 Hash0 = HashPosition( Position0 ); uint32 Hash1 =
			HashPosition( Position1 ); uint32 Hash =
			Murmur32( { Hash0, Hash1 } );

			//Note that the concurrent version is used to add elements here.

			Add_Concurrent. EdgeHash.Add_Concurrent( Hash, EdgeIndex );
		});

	constint32 NumDwords = FMath::DivideAndRoundUp( BoundaryEdges.Num(), NumBitsPerDWORD
) ;

	ParallelFor(NumDwords,
		[&]( int32 DwordIndex ) {

			constint32 NumIndexes = Indexes.Num();
			constint32 NumBits = FMath::Min( NumBitsPerDWORD, NumIndexes - DwordIndex *
NumBitsPerDWORD );
```

```
                uint32 Mask =1; uint32
                Dword =0;
                for( int32 BitIndex =0; BitIndex < NumBits; BitIndex++, Mask <<=1) {

                        //Compute edge indices.
                        int32 EdgeIndex = DwordIndex * NumBitsPerDWORD + BitIndex;


                        uint32 VertIndex0 = Indexes[ EdgeIndex ]; uint32 VertIndex1 =
                        Indexes[ Cycle3( EdgeIndex ) ];

                        constFVector& Position0 = Verts[VertIndex0].Position; constFVector&
                        Position1 = Verts[VertIndex1].Position;

                        uint32 Hash0 = HashPosition( Position0 ); uint32 Hash1 =
                        HashPosition( Position1 ); uint32 Hash =
                        Murmur32( { Hash1, Hash0 } );

                        //Find edges that share two vertices and go in opposite directions.
                        /*

                                 /\
                                / \
                               o-<<-o
                               o->>-o
                                 \ /
                                  \/
                        * /
                        uint32 FoundEdge = ~0u;
                        for( uint32 OtherEdgeIndex = EdgeHash.First( Hash ); EdgeHash.IsValid(
OtherEdgeIndex ); OtherEdgeIndex = EdgeHash.Next( OtherEdgeIndex ) )
                        {
                                uint32 OtherVertIndex0 = Indexes[ OtherEdgeIndex ]; uint32 OtherVertIndex1 =
                                Indexes[ Cycle3( OtherEdgeIndex ) ];

                                if( Position0 == Verts[ OtherVertIndex1 ].Position &&
                                        Position1 == Verts[ OtherVertIndex0 ].Position )
                                {
                                        //Find matching edges.
                                        //Hash tables are not deterministically ordered. Find a stable match, not just the
                                        first one. FoundEdge = FMath::Min( FoundEdge, OtherEdgeIndex );

                                }
                        }
                        Shared Edges                    Edge Index              ]       =       F ound E d              ge

                        if(FoundEdge == ~0u) {

                                Dword |= Mask;
                        }
                }

                if( Dword )
                {
                        BoundaryEdges.GetData()[DwordIndex] = Dword;
                }
        });

    //Disconnected set of triangles.
    FDisjointSetDisjointSet( NumTriangles );
```

```
	for( uint32 EdgeIndex =0, Num = SharedEdges.Num(); EdgeIndex < Num; EdgeIndex++ ) {

		uint32 OtherEdgeIndex = SharedEdges[EdgeIndex]; if
		( OtherEdgeIndex != ~0u) {

			//	OtherEdgeIndexis a matchEdgeIndexThe minimum index of .
			//	ThisEdgeIndexis a matchOtherEdgeIndexThe minimum index of .


			uint32 ThisEdgeIndex = SharedEdges[ OtherEdgeIndex ];
			check( ThisEdgeIndex != ~0u); check( ThisEdgeIndex <= EdgeIndex );



			if(EdgeIndex > ThisEdgeIndex ) {

				//Previous element points toOtherEdgeIndex
				SharedEdges[EdgeIndex] = ~0u;

			}
			else		if(		Edge Index				>	Other Edge Index				)
			{
				//Test again.
				DisjointSet.UnionSequential( EdgeIndex /3, OtherEdgeIndex /3);

			}
		}
	}

	uint32 BoundaryTime = FPlatformTime::Cycles();
	UE_LOG( LogStaticMesh, Log, TEXT("Boundary [%.2fs], tris: %i, UVs %i%s"),
FPlatformTime::ToMilliseconds( BoundaryTime - Time0 ) /1000.0f, Indexes.Num() /3, NumTexCoords,
bHasColors ? TEXT(", Color") : TEXT("") );

	LOG_CRC( SharedEdges );

	//Triangular division.
	FGraphPartitionerPartitioner( NumTriangles );

	{
		TRACE_CPUPROFILER_EVENT_SCOPE(Nanite::Build::PartitionGraph);

		//Get the center of the triangle.
		autoGetCenter = [ &Verts, &Indexes ]( uint32 TriIndex ) {

			FVector Center; Center = Verts[ Indexes[ TriIndex *3+0 ] ].Position; Center +=
			Verts[ Indexes[ TriIndex *3+1 ] ].Position; Center += Verts[ Indexes[ TriIndex *
			3+2 ] ].Position;returnCenter * (1.0f/3.0f);



		};
		//Build a location connection.
		Partitioner.BuildLocalityLinks(DisjointSet, MeshBounds, GetCenter);

		auto* RESTRICT Graph = Partitioner.NewGraph(NumTriangles *3);

		//Process partition data.
		for( uint32 i =0; i < NumTriangles; i++ ) {

			Graph->AdjacencyOffset[i] = Graph->Adjacency.Num();

			uint32 TriIndex = Partitioner.Indexes[i];
```

```
                for(intk =0; k <3; k++ ) {

                    uint32 EdgeIndex = SharedEdges[3* TriIndex + k ];
                    //Add adjacent edges.

                    if( EdgeIndex != ~0u) {

                        Partitioner.AddAdjacency(Graph, EdgeIndex /3,4*65);
                    }
                }

                //Added location connection.

                Partitioner.AddLocalityLinks(Graph, TriIndex,1);
            }
            G raph -> A djacency O ffset [                                                Number of Triangles                    ]       =

            //Precise divisionCluster.
            Partitioner.PartitionStrict(Graph, FCluster::ClusterSize -4, FCluster::ClusterSize,true);

            check( Partitioner.Ranges.Num() );

            LOG_CRC(Partitioner.Ranges);
        }
        //Calculate the idealClusterquantity.

        constuint32 OptimalNumClusters = FMath::DivideAndRoundUp< int32 >( Indexes.Num(), FCluster::ClusterSize *
3);

uint32 ClusterTime = FPlatformTime::Cycles();
UE_LOG( LogStaticMesh, Log, TEXT("Clustering [%.2fs]. Ratio: %f"),
FPlatformTime::ToMilliseconds( ClusterTime - BoundaryTime ) /1000.0f, (float
)Partitioner.Ranges.Num() / OptimalNumClusters );

        constuint32 BaseCluster = Clusters.Num();
        Clusters.AddDefaulted( Partitioner.Ranges.Num() );
        //Author's Note:3on2eIs the thread reversed?
                                                    ?
        const boolbSingleThreaded = Partitioner.Ranges.Num() >32; {

            TRACE_CPUPROFILER_EVENT_SCOPE(Nanite::Build::BuildClusters); //Parallel build
            Cluster.
            ParallelFor( Partitioner.Ranges.Num(),
                [&]( int32 Index ) {

                    auto& Range = Partitioner.Ranges[ Index ];

                    //Create a singleClusterExamples.
                    Clusters[ BaseCluster + Index ] = FCluster( Verts,
                                                                Indexes,
                                                                MaterialIndexes,
                                                                BoundaryEdges, Range.Begin,
Range.End, Partitioner.Indexes, NumTexCoords, bHasColors );

                    //A negative number indicates that it is a leaf.

                    Clusters[ BaseCluster + Index ].EdgeLength *=-1.0f; bSingleThreaded);
                },
        }
```

```
        uint32 LeavesTime = FPlatformTime::Cycles();
        UE_LOG( LogStaticMesh, Log, TEXT("Leaves [%.2fs]"), FPlatformTime::ToMilliseconds( LeavesTime -
ClusterTime ) /1000.0f); }
```

6.4.2.4 FGraphPartitioner

The code in the previous section uses FGraphPartitioner when processing Cluster. Let's analyze its code:

```
// Engine\Source\Developer\NaniteBuilder\Private\GraphPartitioner.h

(......)

//CitedmetisThird-party open source libraries.
#include"metis.h"

(......)

// ClusterPartition diagram
classFGraphPartitioner {

public:
        //Graph data.
        structFGraphData
        {
                int32           Offset;     //  Index displacement.
                int32           Num;        //  quantity.

                TArray<     idx_t     >     Adjacency;//Neighbor list
                TArray<     idx_t     >     AdjacencyCost;//Neighbor weight list
                TArray<     idx_t     >     AdjacencyOffset;//Adjacent Edge Displacement List
        };

        //The range is [Begin,    End]
        structFRange
        {
                uint32          Begin;
                uint32          End;

                booloperator<(constFRange & Other)const{returnBegin < Other.Begin; } }; TArray<
                        FRange
        > TArray<       uint32      Ranges;
        >                           Indexes;

public:
                        FGraphPartitioner( uint32 InNumElements );

        //Construct a new subgraph data instance.
        FGraphData*         NewGraph( uint32 NumAdjacency )const;

        //Add adjacent edges.
        void                AddAdjacency(FGraphData* Graph, uint32 AdjIndex,idx_tCost );
        //Added location connection.
        void                AddLocalityLinks(FGraphData* Graph, uint32 Index,idx_tCost );
```

```
        //Build a location connection.
        template<typename FGetCenter> void
                        BuildLocalityLinks( FDisjointSet& DisjointSet,constFBounds& Bounds, GetCenter );
FGetCenter&

        //DivisionCluster.
        void            Partition(FGraphData* Graph, int32 InMinPartitionSize, int32
InMaxPartitionSize );
        //Precise divisionCluster.
        void            PartitionStrict(FGraphData* Graph, int32 InMinPartitionSize, int32
InMaxPartitionSize,boolbThreaded );

private:
        //Bisect the subgraph.
        void
                        BisectGraph(FGraphData* Graph, FGraphData* ChildGraphs[2] );
        //Recursively bisect subgraphs.
        void            RecursiveBisectGraph( FGraphData* Graph );

        uint32          NumElements;
        int32           MinPartitionSize    =  0;
        int32           MaxPartitionSize    =  0;

        // ClusterQuantity. Uses atomics to support multi-threaded reading and writing
        TAtomic< uint32 > NumPartitions;

        TArray<     idx_t     >         PartitionIDs;
        TArray<     int32     >         SwappedWith;
        TArray<    uint32    >       SortedTo;

        //Location connection.
        TMultiMap< uint32, uint32 >             LocalityLinks;
};

(.....)

// Engine\Source\Developer\NaniteBuilder\Private\GraphPartitioner.cpp

(.....)

//Bisect the grid.
void FGraphPartitioner::BisectGraph(FGraphData* Graph, FGraphData* ChildGraphs[2] ) {

        ChildGraphs[0]      =  nullptr;
        ChildGraphs[1]      =  nullptr;

        //Added partition callback.
        autoAddPartition =
                [ this ]( int32 Offset, int32 Num ) {

                        FRange& Range = Ranges[ NumPartitions++ ];
                        Range.Begin         = Offset;
                        Range.End       = Offset + Num;
                };

        // ifGraph      If the number of partitions does not exceed the limit, add them directlythismiddle.
        if(Graph->Num <= MaxPartitionSize) {

                AddPartition( Graph->Offset, Graph->Num );
```

```
            return;
        }

        //Calculate expected partition size.
        constint32 TargetPartitionSize = (MinPartitionSize + MaxPartitionSize) /2; constint32 TargetNumPartitions =
        FMath::Max(2, FMath::DivideAndRoundNearest( Graph-
> Num, TargetPartitionSize ) );

        check( Graph->AdjacencyOffset.Num() == Graph->Num +1);

        idx_tNumConstraints =1; idx_t
        NumParts =2;idx_t EdgesCut =0;


        real_tPartitionWeights[] = {
            float( TargetNumPartitions /2) / TargetNumPartitions,
            1.0f-float( TargetNumPartitions /2) / TargetNumPartitions
        };

        //set upMetisDefault operating parameters for the library.
        idx_tOptions[ METIS_NOPTIONS ];
        METIS_SetDefaultOptions( Options                ) ;

        //Loose tolerances are allowed at high levels, and strict balancing is not important until closer to the partition size.
        boolbLoose = TargetNumPartitions >=128|| MaxPartitionSize / MinPartitionSize >1; boolbSlow = Graph->Num <
        4096;

        Options[ METIS_OPTION_UFACTOR ] = bLoose ?200:1;
        //Options[ METIS_OPTION_NCUTS ] = Graph->Num < 1024 ? 8 : ( Graph->Num < 4096 ? 4 : 1
) ;
        //Options[ METIS_OPTION_NCUTS ] = bSlow ? 4 : 1; // Options[
        METIS_OPTION_NITER   ] =  bSlow  ?  20  :  10;  //  Options[
        METIS_OPTION_IPTYPE ] = METIS_IPTYPE_RANDOM; // Options[
        METIS_OPTION_MINCONN ] = 1;

        //CallMetisRecursive partitioning of .
        intr = METIS_PartGraphRecursive(
            &Graph->Num,
            &NumConstraints,                    // number of balancing constraints
            Graph->AdjacencyOffset.GetData(), Graph-
            >Adjacency.GetData(),               NULL,
            NULL,                               //Vert weights
                                                // Vert sizes for computing the total communication
volume
            Graph->AdjacencyCost.GetData(),             // Edge weights
            &NumParts,
            PartitionWeights,                   // Target partition weight
            NULL,                               // Allowed load imbalance tolerance
            Options,
            &EdgesCut,
            PartitionIDs.GetData()
                                + Graph->Offset
        ) ;

        //confirmMetisThe result of recursive partitioning is valid.
        if( ensureAlways( r == METIS_OK ) ) {

            //Splits the array in place.
            //Both sides remain sorted, but the order is reversed.
```

```
int32 Front = Graph->Offset;
int32 Back = Graph->Offset + Graph->Num -1; while( Front
<= Back ) {

    while( Front <= Back && PartitionIDs[ Front ] ==0) {

        SwappedWith[Front] = Front; Front++;

    }
    while(          Front          < =     Back          & &     Partition IDs                    [
    {
        SwappedWith[ Back ] = Back; Back--;

    }

    if( Front < Back ) {

        Swap( Indexes[ Front ], Indexes[ Back ] );

        SwappedWith[Front] = Back;
        SwappedWith[Back] = Front; Front++;
        Back--;

    }
}

int32 Split = Front;

int32 Num[2];
Num[0] = Split - Graph->Offset;
Num[1] = Graph->Offset + Graph->Num - Split;

check( Num[0] >1);
check( Num[1] >1);

//If the partition size of the two child nodes does not exceed the limit, add them directly.
if( Num[0] <= MaxPartitionSize && Num[1] <= MaxPartitionSize ) {

    AddPartition(    Graph->Offset,        Num[0]   ) ;
    AddPartition(    Split,                Num[1]   ) ;
}
else
{
    //Create two child node instances.
    for( int32 i =0; i <2; i++ ) {

        ChildGraphs[i] = new FGraphData; ChildGraphs[i]->Adjacency.Reserve( Graph-
        >Adjacency.Num() >>1); ChildGraphs[i]->AdjacencyCost.Reserve( Graph->Adjacency.Num() >>1
        ); ChildGraphs[i]->AdjacencyOffset.Reserve( Num[i] +1); ChildGraphs[i]->Num = Num[i];



    }

    ChildGraphs[0]->Offset      =Graph->Offset;
    ChildGraphs[1]->Offset      = Split;

    //Traverse all sub-partitions andGraphThe adjacent edge ofChildGraphs[0]or
    ChildGraphs[1] for( int32 i =0; i < Graph->Num; i++ )
```

```
                {
                    //The code here is a bittrick:likei<=ChildGraphs[0]->NumThen getChildGraphs[0],Otherwise get
ChildGraphs[1].
                    FGraphData* ChildGraph = ChildGraphs[ i >= ChildGraphs[0]->Num ];

                    ChildGraph->AdjacencyOffset.Add( ChildGraph->Adjacency.Num() );

                    int32 OrgIndex = SwappedWith[ Graph->Offset + i ] - Graph->Offset; for(idx_tAdjIndex =
                    Graph->AdjacencyOffset[ OrgIndex ]; AdjIndex <
Graph->AdjacencyOffset[ OrgIndex +1]; AdjIndex++ )
                    {
                        idx_tAdj              = Graph->Adjacency[AdjIndex];
                        idx_tAdjCost = Graph->AdjacencyCost[AdjIndex];

                        // Remap to child
                        Adj = SwappedWith[ Graph->Offset + Adj ] - ChildGraph->Offset;

                        // Edge connects to node in this graph if(0<= Adj &&
                        Adj < ChildGraph->Num ) {

                            ChildGraph->Adjacency.Add(   Adj          ) ;
                            ChildGraph->AdjacencyCost.Add(            AdjCost     ) ;
                        }
                    }
                }                                                   Child Graphs0] -
                > Add (
                Child Graphs0] -> A djacency . N um ( ) ) ;
            }                                                       Child Graphs1] -
        }           > Add (
}                   Child Graphs1] -> A djacency . N um ( ) ) ;

//    Precise division
void FGraphPartitioner::PartitionStrict(FGraphData* Graph, int32 InMinPartitionSize, int32 InMaxPartitionSize,bool
bThreaded ) {

    MinPartitionSize        =   InMinPartitionSize;
    MaxPartitionSize        =  InMaxPartitionSize;

                                                NumElements    ) ;
    PartitionIDs.AddUninitialized( SwappedWNituhm.AEdldeUmneintitsiali)z ;ed(

    // Adding to atomically so size big enough to not need to grow.
    int32 NumPartitionsExpected = FMath::DivideAndRoundUp( Graph->Num, MinPartitionSize );
    Ranges.AddUninitialized( NumPartitionsExpected *2); NumPartitions =0;


    //Use multithreading.
    if( bThreaded && NumPartitionsExpected >4) {

        externCORE_API int32 GUseNewTaskBackend;
        //Use background threads.

        if(GUseNewTaskBackend) {

            //Local Work Queue
            TLocalWorkQueue<FGraphData>LocalWork(Graph); //HereSelf
            refer toLambdaThe function itself.
            LocalWork.Run(MakeYCombinator([this, &LocalWork](autoSelf, FGraphData* Graph)
->  void
```

```
                {
                        FGraphData*     ChildGraphs[2];
                        //Divide equally.
                        BisectGraph( Graph, ChildGraphs ); delete
                        Graph;


                        if( ChildGraphs[0] && ChildGraphs[1] ) {

                                //Processing1Subnodes
                                //New worker threads are added only if the remaining work is
                                large enough if(ChildGraphs[0]->Num >256) {


                                        LocalWork.AddTask(ChildGraphs[0]);
                                        LocalWork.AddWorkers(1);
                                }
                                else                    //              Otherwise, call recursively.
                                {
                                        Self(ChildGraphs[0]);
                                }

                                //Processing2Subnodes
                                Self(ChildGraphs[1]);

                        }
                }));
        }
        //          Not a background thread.        Use traditionalT ask GraphMission system.
        else
        {
                constENamedThreads::Type DesiredThread = IsInGameThread() ?
ENamedThreads::AnyThread : ENamedThreads::AnyBackgroundThreadNormalTask;

                //Build tasks.
                classFBuildTask
                {
                public:
                        FBuildTask( FGraphPartitioner* InPartitioner, FGraphData* InGraph,
ENamedThreads::Type     InDesiredThread)
                                : Partitioner(InPartitioner),
                                Graph(InGraph           )
                                , DesiredThread(        InDesiredThread )
                        {}

                        void DoTask( ENamedThreads::Type CurrentThread,constFGraphEventRef&
MyCompletionEvent       )
                        {
                                FGraphData* ChildGraphs[2]; Partitioner->BisectGraph( Graph,
                                ChildGraphs ); delete Graph;


                                if( ChildGraphs[0] && ChildGraphs[1] ) {

                                        if( ChildGraphs[0]->Num >256) {

                                                FGraphEventRef Task = TGraphTask< FBuildTask
> ::CreateTask().ConstructAndDispatchWhenReady( Partitioner, ChildGraphs[0], DesiredThread);

                                                MyCompletionEvent->DontCompleteUntil(Task);
                                        }
```

```
                                else
                                {
                                        FBuildTask(     Partitioner,     ChildGraphs[0],
DesiredThread).DoTask(CurrentThread, MyCompletionEvent);
                                }

                                FBuildTask(Partitioner, ChildGraphs[1],
CurrentThread,                  DesiredThread).DoTask( MyCompletionEvent );
                        }
                }

                staticFORCEINLINE TStatIdGetStatId() {

                        RETURN_QUICK_DECLARE_CYCLE_STAT(FBuildTask,
STATGROUP_ThreadPoolAsyncTasks);
                }

                staticFORCEINLINE ESubsequentsMode::Type        GetSubsequentsMode()    {
returnESubsequentsMode::TrackSubsequents; }

                FORCEINLINE ENamedThreads::TypeGetDesiredThread()const {

                        returnDesiredThread;
                }

        private:
                FGraphPartitioner*      Partitioner;
                FGraphData*             Graph;
                ENamedThreads::Type     DesiredThread;
        };

        FGraphEventRef BuildTask = TGraphTask< FBuildTask >::CreateTask( nullptr
) . ConstructAndDispatchWhenReady( this, Graph, DesiredThread);
        FTaskGraphInterface::Get().WaitUntilTaskCompletes( BuildTask );
    }
}
else
{
    RecursiveBisectGraph(Graph);
}

Ranges.SetNum(NumPartitions);

if( bThreaded ) {

    // Force a deterministic order
    Ranges.Sort();
}

PartitionIDs.Empty();
SwappedWith.Empty();
}
```

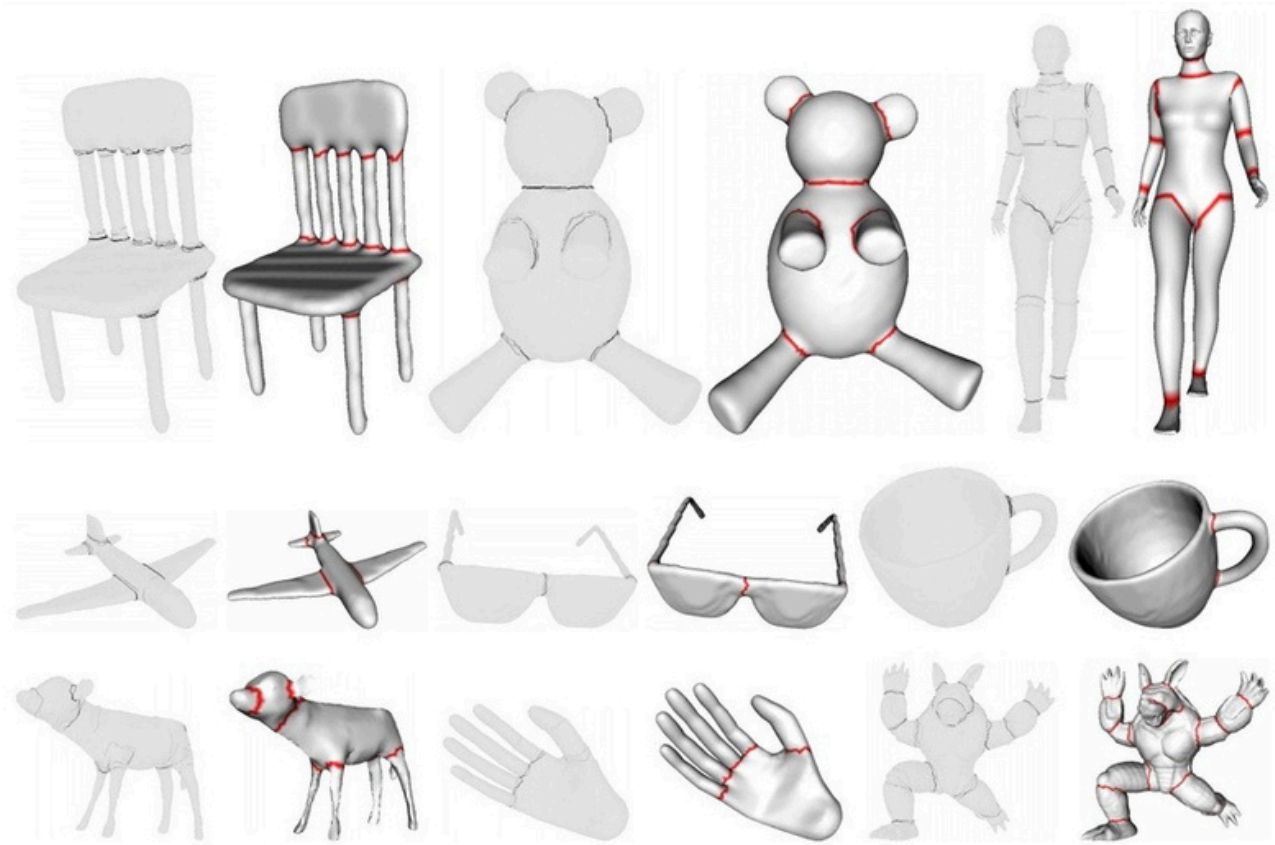
Regarding the meshing of Nanite, the following explanation is added here:

- A lot of parallel processing is used when building Nanite, including but not limited to processing edge hashes, detecting shared edges and boundary edges, building Clusters,

dividing grids, generating Imposters, etc., to shorten the construction time of Nanite data. When
- dividing the grid, GUseNewTaskBackend is used to decide whether to enable new background task parallel processing or traditional TaskGraph. The new background task system is a feature added in UE5 and is more lightweight and concise.
- Several key interfaces of the third-party open source library METIS are used when dividing the grid: METIS_SetDefaultOptions, METIS_PartGraphKway, and METIS_PartGraphRecursive.

> **METIS**is a set of serial programs for partitioning graphs, partitioning finite element meshes and generating fill reduction orders for sparse matrices. The algorithms implemented in METIS are based on multi-level recursive bisection, multi-level k-way and multi-constrained partitioning schemes developed by Karypis Laboratory. Its key features are:
>
> - **Provides high-quality partitioning.**Partitions produced by METIS are consistently better than those produced by other widely used algorithms. Partitions produced by METIS are consistently 10% to 50% better than those produced by spectral partitioning algorithms. **Exceptionally fast**
> - **processing speed.**Extensive practice shows that METIS is one to two orders of magnitude faster than other widely used partitioning algorithms. On current workstations and PCs, graphs with millions of vertices can be partitioned into 256 parts in a few seconds.
>
> - **The generated results have a low fill-in ratio.**The reduced fill-in orderings produced by METIS significantly outperform other widely used algorithms, including multiple minimum degree. For many classes of problems arising in scientific computing and linear programming, METIS is able to reduce the storage and computational requirements of sparse matrix factorization by an order of magnitude. Unlike multiple minimum degree methods, the elimination trees generated by METIS are suitable for parallel direct factorization. In addition, METIS is able to compute these orderings very quickly. On current workstations and PCs,
>   Matrices with millions of rows can be reordered in seconds.
>
> It also has a parallelized version**ParMETIS**.For details, please refer to the official description: Family of Graph and Hypergraph Partitioning Software.

- The steps, details and logic of meshing are relatively complicated, but the author believes that Nanite's ideas and intentions are similar to those in the manuscriptMETIS Three Phases Coarsening Partitioning Uncoarseningand the paperLearning Boundary Edges for 3D-Mesh Segmentation, so the algorithm and process of meshing are explained accordingly.
  When a model needs to be divided into several parts, it can be divided manually using**the Princeton segmentation benchmark**,or it can be divided automatically with the help of certain algorithms (see the figure below).

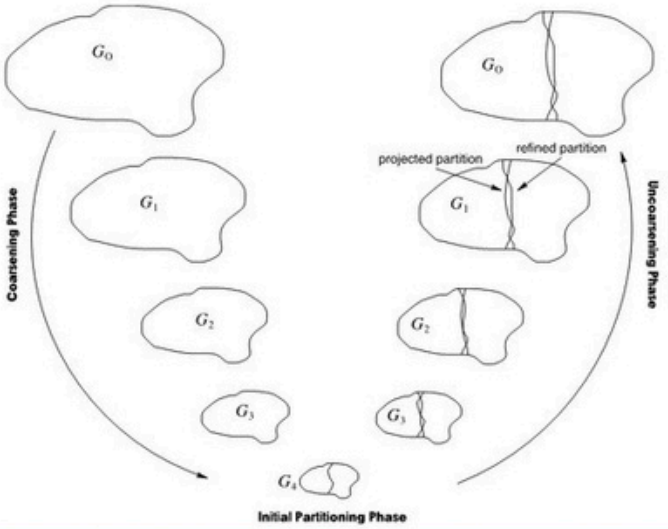


There are multiple pairs of graphs in the figure above. The left side of each pair of graphs is manually divided based on the Princeton Partition Principle (the dark lines represent the manually divided edges), and the right side of the pair of graphs is automatically divided by the algorithm (the red lines represent the boundaries). It can be seen that the automatic division algorithm can highly match the manual division.

Automatic segmentation algorithms include methods that combine deep learning and vision, as well as traditional mathematical algorithms like METIS. The METIS segmentation algorithm has three stages: coarsening, partitioning, and uncoarsening.

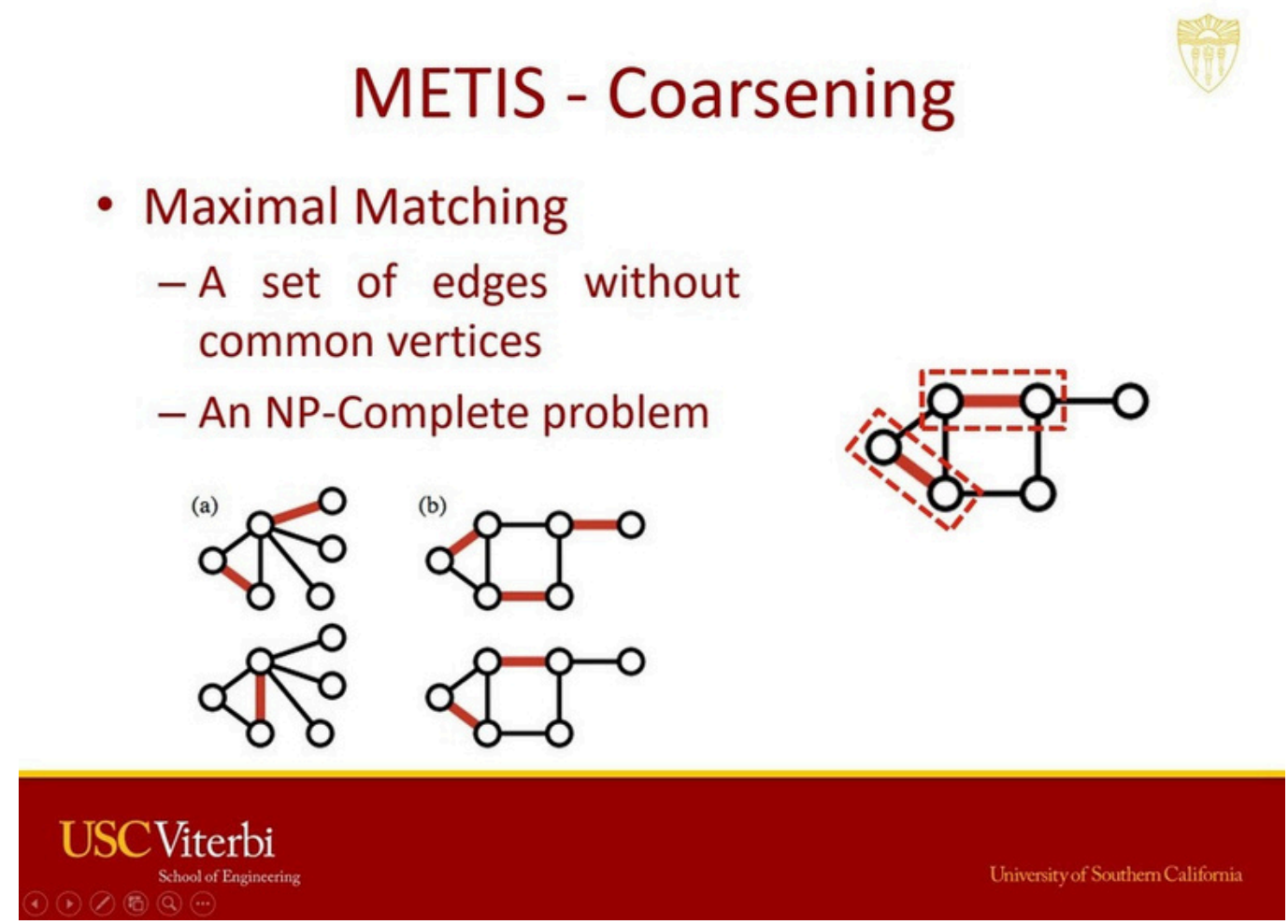
METIS

- Three Phases
  - Coarsening
  - Partitioning
  - Uncoarsening



G. Karypis, V. Kumar, "A fast and high quality multilevel scheme for partitioning irregular graphs," *International Conference on Parallel Processing, 1995.*

In the Coarsening phase, the maximum matching is to find the set of edges without common vertices, which is an NP-complete problem in terms of search complexity.

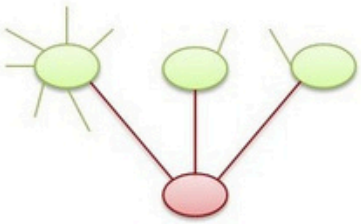


Coarsening has an NP-complete problem when matching the maximum edge. For example, group a obviously does not have the largest number of non-shared vertex edges, but group b does.

In the Partitioning phase, two steps are required. The first step is to randomly select a root, and the second step is a breadth first search (BFS) to include vertices that can obtain fewer cut edges.

METIS - Partitioning

- Two Steps
  - Randomly Choose a ***root***
  - BFS to include the vertex leading less **edge-cuts**



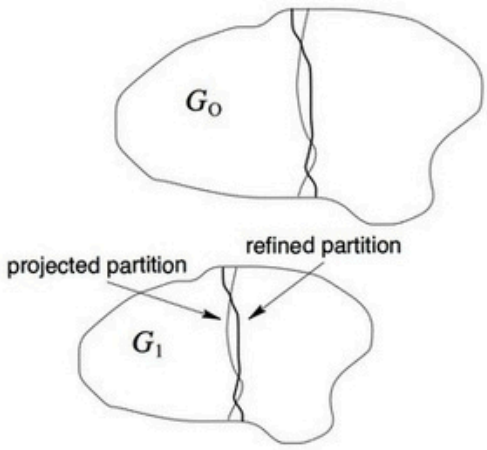
USC Viterbi School of Engineering

University of Southern California

The key idea in the Uncoarsening phase is that each parent node contains a set of child nodes, and the cut edges are reduced by moving vertices from one partition to another.

METIS - Uncoarsening

- Key Idea
  - Each super-node comprises a set of nodes
  - Decrease the edge-cuts by moving a vertex to one partition to another



USC Viterbi School of Engineering
University of Southern California

6.4.2.5 BuildDAG

```
// Engine\Source\Developer\NaniteBuilder\Private\ClusterDAG.cpp

//BuildClusterA directed acyclic graph of .
void BuildDAG( TArray< FClusterGroup >& Groups, TArray< FCluster >& Clusters, uint32 ClusterRangeStart, uint32 ClusterRangeNum, uint32 MeshIndex, FBounds& MeshBounds ) {

	uint32 LevelOffset		= ClusterRangeStart;

	TAtomic< uint32 >	NumClusters(Clusters.Num() );
	uint32			NumExternalEdges =0;

	boolbFirstLevel =true;

	while(true)
	{
		TArrayView< FCluster >LevelClusters( &Clusters[LevelOffset], bFirstLevel ? ClusterRangeNum : (Clusters.Num() - LevelOffset) );
		bFirstLevel =false;

		for( FCluster& Cluster : LevelClusters ) {

			NumExternalEdges	+ = Cluster.NumExternalEdges;
			MeshBounds		+ = Cluster.Bounds;
		}

		if(LevelClusters.Num() <2)
```

```
                break;

            //If the levelClusterIf the number is less than the maximum number of groups, add it to the group list directly.
            if( LevelClusters.Num() <= MaxGroupSize ) {

                TArray< uint32, TInlineAllocator< MaxGroupSize > > Children;

                uint32 MaxParents =0;
                for( FCluster& Cluster : LevelClusters ) {

                    MaxParents += FMath::DivideAndRoundUp< uint32 >( Cluster.Indexes.Num(),
FCluster::ClusterSize *6);
                    Children.Add( LevelOffset++ );
                }

                LevelOffset = Clusters.Num();
                Clusters.AddDefaulted(          MaxParents     ) ;
                Groups.AddDefaulted(1           ) ;

                //UsageDAGSubtract vertices and faces and add them to the corresponding group.
                DAGReduce( Groups, Clusters, NumClusters, Children, Groups.Num() -1,
MeshIndex    ) ;

                //Correct num to atomic count
                Clusters.SetNum(NumClusters,false);

                continue;
            }
            //This levelClusterAmount greater thanMaxGroupSize,Need to useFGraphPartitionerDivide.
            //External edge structure

            structFExternalEdge
            {


                uint32          ClusterIndex;
                uint32          EdgeIndex;
            };
            //List of external edges.
            TArray< FExternalEdge          >        ExternalEdges;
            FHashTable                               ExternalEdgeHash;
            TAtomic< uint32 >                        ExternalEdgeOffset(0);

            //haveNumExternalEdges, so a non-growing hash table can be allocated.
            ExternalEdges.AddUninitialized( NumExternalEdges ); ExternalEdgeHash.Clear(1<<
            FMath::FloorLog2(NumExternalEdges), NumExternalEdges);

            NumExternalEdges =0;

            //Add edges to the hash table in parallel.
            ParallelFor( LevelClusters.Num(),
                [&]( uint32 ClusterIndex ) {

                    FCluster& Cluster = LevelClusters[ClusterIndex];

                    for( TConstSetBitIterator<> SetBit( Cluster.ExternalEdges ); SetBit;
+ +SetBit     )
                    {
                        uint32 EdgeIndex = SetBit.GetIndex();
```

```
                        uint32 VertIndex0 = Cluster.Indexes[ EdgeIndex ]; uint32 VertIndex1 =
                        Cluster.Indexes[ Cycle3( EdgeIndex ) ];

                        constFVector& Position0 = Cluster.GetPosition( VertIndex0 ); constFVector&
                        Position1 = Cluster.GetPosition( VertIndex1 );

                        uint32 Hash0 = HashPosition( Position0 ); uint32 Hash1 =
                        HashPosition( Position1 ); uint32 Hash =
                        Murmur32( { Hash0, Hash1 } );

                        uint32 ExternalEdgeIndex = ExternalEdgeOffset++; ExternalEdges[ ExternalEdgeIndex ] =
                        { ClusterIndex, EdgeIndex }; ExternalEdgeHash.Add_Concurrent( Hash,
                        ExternalEdgeIndex );
                    }
                });

            check(ExternalEdgeOffset == ExternalEdges.Num() );

            TAtomic< uint32 >NumAdjacency(0);

            //In parallel with otherClusterFind matching edges.
            ParallelFor( LevelClusters.Num(),
                [&]( uint32 ClusterIndex ) {

                    FCluster& Cluster = LevelClusters[ClusterIndex];

                    for( TConstSetBitIterator<> SetBit( Cluster.ExternalEdges ); SetBit;
+ +SetBit     )
                    {
                        uint32 EdgeIndex = SetBit.GetIndex();

                        uint32 VertIndex0 = Cluster.Indexes[ EdgeIndex ]; uint32 VertIndex1 =
                        Cluster.Indexes[ Cycle3( EdgeIndex ) ];

                        constFVector& Position0 = Cluster.GetPosition( VertIndex0 ); constFVector&
                        Position1 = Cluster.GetPosition( VertIndex1 );

                        uint32 Hash0 = HashPosition( Position0 ); uint32 Hash1 =
                        HashPosition( Position1 ); uint32 Hash =
                        Murmur32( { Hash1, Hash0 } );

                        for( uint32 ExternalEdgeIndex = ExternalEdgeHash.First( Hash );
ExternalEdgeHash.IsValid( ExternalEdgeIndex ); ExternalEdgeIndex = ExternalEdgeHash.Next( ExternalEdgeIndex )
                        )
                        {
                            FExternalEdge ExternalEdge = ExternalEdges[ExternalEdgeIndex];

                            FCluster& OtherCluster = LevelClusters[ExternalEdge.ClusterIndex
];

                            if( OtherCluster.ExternalEdges[ ExternalEdge.EdgeIndex ] ) {

                                uint32 OtherVertIndex0 = OtherCluster.Indexes[
ExternalEdge.EdgeIndex          ];
                                uint32 OtherVertIndex1 = OtherCluster.Indexes[ Cycle3(
ExternalEdge.EdgeIndex          ) ];
```

```
                                        if( Position0 == OtherCluster.GetPosition( OtherVertIndex1 )
&&
                                            Position1 == OtherCluster.GetPosition( OtherVertIndex0 ) )
                                        {
                                            //When a matching edge is found, increment its count.
                                            Cluster.AdjacentClusters.FindOrAdd(
ExternalEdge.ClusterIndex,0)++;

                                            // Can't break or a triple edge might be non-
deterministically        connected.
                                            // Need to find all matching, not just first.
                                        }
                                    }
                                }
                            }
                            N um A djacency                         + =       Cluster . Additional Cluster                                ust

                            //Enforces a deterministic ordering of adjacent edges.
                            Cluster.AdjacentClusters.KeySort(
                                [ &LevelClusters ]( uint32 A, uint32 B ) {

                                    returnLevelClusters[A].GUID < LevelClusters[B].GUID;
                                } );
                        });

                //DiscontinuousClusterA collection of.
                FDisjointSetDisjointSet( LevelClusters.Num() );

                for( uint32 ClusterIndex =0; ClusterIndex < (uint32)LevelClusters.Num(); ClusterIndex++ )

                {
                    for(auto& Pair: LevelClusters[ClusterIndex].AdjacentClusters) {

                        uint32 OtherClusterIndex = Pair.Key;

                        uint32 Count = LevelClusters[ OtherClusterIndex
].AdjacentClusters.FindChecked( ClusterIndex );
                        check( Count == Pair.Value );

                        if(ClusterIndex > OtherClusterIndex ) {

                            DisjointSet.UnionSequential(ClusterIndex, OtherClusterIndex);
                        }
                    }
                }

                //Divider.
                FGraphPartitionerPartitioner( LevelClusters.Num() );

                //Sort to enforce deterministic order.
                {
                    TArray< uint32 > SortedIndexes;
                    SortedIndexes.AddUninitialized( RadixSort3P2a( rStoitritoendeIrn.IdnedxexSe.Gs.eNtDumat(a)(),
                                                                   Partitioner.Indexes.GetData(),
Partitioner.Indexes.Num(),
                        [&]( uint32 Index ) {

                            returnLevelClusters[ Index ].GUID;
```

```
                } );
            Swap(Partitioner.Indexes, SortedIndexes);
        }

        autoGetCenter = [&]( uint32 Index ) {

            FBounds& Bounds = LevelClusters[ Index ].Bounds; return0.5f*
            (Bounds.Min + Bounds.Max);
        };
        //Build a location connection.
        Partitioner.BuildLocalityLinks(DisjointSet, MeshBounds, GetCenter);


        auto* RESTRICT Graph = Partitioner.NewGraph(NumAdjacency);

        //Traverse all levelsCluster,Then traverse all theCluster,Add adjacent edges and positional connections.
        for( int32 i =0; i < LevelClusters.Num(); i++ ) {

            Graph->AdjacencyOffset[i] = Graph->Adjacency.Num();

            uint32 ClusterIndex = Partitioner.Indexes[i];

            for(auto& Pair: LevelClusters[ClusterIndex].AdjacentClusters) {

                uint32 OtherClusterIndex = Pair.Key; uint32
                NumSharedEdges = Pair.Value;

                constauto& Cluster0 = Clusters[LevelOffset + ClusterIndex]; constauto& Cluster1 =
                Clusters[LevelOffset + OtherClusterIndex];

                boolbSiblings = Cluster0.GroupIndex != MAX_uint32 && Cluster0.GroupIndex
== Cluster1.GroupIndex;

                Partitioner.AddAdjacency( Graph, OtherClusterIndex, NumSharedEdges * (
bSiblings ?1:16) +4);
            }

            Partitioner.AddLocalityLinks(Graph, ClusterIndex,1);
        }
        G raph -> A djacency O ffset [                                    Graph -> Number                    ]      =      G

        LOG_CRC(    Graph->Adjacency );
        LOG_CRC(    Graph->AdjacencyCost );
        LOG_CRC(    Graph->AdjacencyOffset          ) ;

        //Strict zoning.
        Partitioner.PartitionStrict(Graph, MinGroupSize, MaxGroupSize,true);

        LOG_CRC(Partitioner.Ranges);

        //Calculate the maximum number of parents.
        uint32 MaxParents =0;
        for(auto& Range : Partitioner.Ranges ) {

            uint32 NumParentIndexes =0;
            for( uint32 i = Range.Begin; i < Range.End; i++ ) {

                // Global indexing is needed in Reduce()
                Partitioner.Indexes[i] += LevelOffset;
```

```
                NumParentIndexes += Clusters[ Partitioner.Indexes[i] ].Indexes.Num();
            }
            Max Parts                    + =    FM ath : : D ivide A nd R ound U p (
* 6 ) ;
        }

        LevelOffset = Clusters.Num();

        Clusters.AddDefaulted( MaxParents );
        Groups.AddDefaulted( Partitioner.Ranges.Num() );

        //Execute in parallelDAGReduce the surface and reduce the mold.
        ParallelFor( Partitioner.Ranges.Num(),
            [&]( int32 PartitionIndex ) {

                auto& Range = Partitioner.Ranges[PartitionIndex];

                TArrayView< uint32 > Children( &Partitioner.Indexes[ Range.Begin ],
Range.End - Range.Begin );
                uint32 ClusterGroupIndex = PartitionIndex + Groups.Num() -
Partitioner.Ranges.Num();

                DAGReduce( Groups, Clusters, NumClusters, Children, ClusterGroupIndex,
MeshIndex ) ;
            });

        //Correct num to atomic count
        Clusters.SetNum(NumClusters,false);
    }

    //Maximum output root node.
    uint32 RootIndex = LevelOffset; FClusterGroup
    RootClusterGroup; RootClusterGroup.Children.Add( RootIndex );
    RootClusterGroup.Bounds = Clusters[ RootIndex ].SphereBounds;
    RootClusterGroup.LODBounds=FSphere(0);
    RootClusterGroup.MaxParentLODError = 1e10f;
    RootClusterGroup.MinLODError = - 1.0f;
    RootClusterGroup.MipLevel = Clusters[RootIndex].MipLevel +1;
    RootClusterGroup.MeshIndex = MeshIndex; Clusters[ RootIndex ].GroupIndex =
    Groups.Num(); Groups.Add( RootClusterGroup );


}
```

DAGReduce was executed several times above, and its implementation is briefly analyzed:

```
static void DAGReduce(TArray< FClusterGroup >& Groups, TArray< FCluster >& Clusters, TAtomic< uint32 >&
NumClusters, TArrayView< uint32 > Children, int32 GroupIndex, uint32 MeshIndex )

{
    check( GroupIndex >=0);

    //mergeCluster.
    TArray<constFCluster*, TInlineAllocator<16> > MergeList; for( int32 Child :
    Children ) {

        MergeList.Add( &Clusters[ Child ] );
```

```
        }
        //Forced order.
        MergeList.Sort(

                [](constFCluster& A,constFCluster& B ) {

                        returnA.GUID < B.GUID;
                } );

        FClusterMerged( MergeList );

        int32 NumParents = FMath::DivideAndRoundUp< int32 >( Merged.Indexes.Num(),
FCluster::ClusterSize *6);
        int32 ParentStart =0; int32
        ParentEnd =0;

        floatParentMaxLODError=0.0f;

        //NoticeTargetClusterSizeStep length -2.
        for(int32 TargetClusterSize = FCluster::ClusterSize -2; TargetClusterSize > FCluster::ClusterSize /2;
TargetClusterSize -=2)
        {
                int32 TargetNumTris = NumParents * TargetClusterSize;

                //Simplified, it will return the maximum parent nodeLODerror.
                ParentMaxLODError = Merged.Simplify( TargetNumTris );

                //Split
                if( NumParents ==1) {

                        ParentEnd = (NumClusters += NumParents); ParentStart
                        = ParentEnd - NumParents;

                        Clusters[ ParentStart ] = Merged;
                        Clusters[ ParentStart ].Bound(); break;

                }
                else
                {
                        FGraphPartitioner Partitioner( Merged.Indexes.Num() /3);
                        Merged.Split( Partitioner );

                        if( Partitioner.Ranges.Num() <= NumParents ) {

                                NumParents = Partitioner.Ranges.Num(); ParentEnd =
                                (NumClusters += NumParents); ParentStart = ParentEnd -
                                NumParents;

                                int32 Parent = ParentStart;
                                for(auto& Range : Partitioner.Ranges ) {

                                        Clusters[ Parent ] = FCluster( Merged, Range.Begin, Range.End,
Partitioner.Indexes );
                                        Parent++;
                                }

                                break;
                        }
```

```
            }
        }

        TArray< FSphere, TInlineAllocator<32> > Children_LODBounds; TArray< FSphere,
        TInlineAllocator<32> > Children_SphereBounds;
        //Enforce monotonic nesting (monotonic
        nesting). floatChildMinLODError = MAX_flt; for
        ( int32 Child : Children ) {


            boolbLeaf = Clusters[Child].EdgeLength <0.0f; floatLODError =
            Clusters[Child].LODError;

            Children_LODBounds.Add( Clusters[ Child ].LODBounds );
            Children_SphereBounds.Add( Clusters[ Child ].SphereBounds ); ChildMinLODError =
            FMath::Min( ChildMinLODError, bLeaf ?-1.0f: LODError ); ParentMaxLODError =
            FMath::Max( ParentMaxLODError, LODError );

            Clusters[ Child ].GroupIndex = GroupIndex; Groups[ GroupIndex ].Children.Add( Child );
            check( Groups[ GroupIndex ].Children.Num() <= MAX_CLUSTERS_PER_GROUP_TARGET );

        }

        FSphere         ParentLODBounds( Children_LODBounds.GetData(), Children_LODBounds.Num() ); ParentBounds
        FSphere         (Children_SphereBounds.GetData(), Children_SphereBounds.Num()
    ) ;

        //Force parent nodes to have the sameLODdata, they depend on each other.

        for( int32 Parent = ParentStart; Parent < ParentEnd; Parent++ ) {

            Clusters        Parent ].LODBounds                      =ParentLODBounds;
            Clusters        Parent    ].LODError                        = ParentMaxLODError;
            Clusters[ Parent ].GeneratingGroupIndex = GroupIndex;
        }

        Groups  Groups[ GroupIndex ].Bounds                         = ParentBounds;
        Groups[   GroupIndex ].LODBounds                          = ParentLODBounds;
        Groups[ GroupIndex ].MaxChildLODError                   = ChildMinLODError;
                GroupIndex ].MaxParentLODError                    =    ParentMaxLODError;
                                                                  =Merged.MipLevel  -1;  =
                                                                  MeshIndex;
    }
```

6.4.2.6 BuildCoarseRepresentation

BuildCoarseRepresentation builds a rough representation of the mesh based on the input Cluster list and Cluster group list, and outputs the corresponding vertex, index, Section and other data:

```
static void         BuildCoarseRepresentation
        const    (TArray<FClusterGroup>&Groups,
        TArray<FStaticMeshBuildVertex>&Verts,
        TArray<uint32>&                 Indexes,          Verts,
        TArray<FStaticMeshSection,
        uint32&    NumTexCoords,    uint32  TInlineAllocator<1>>&          Sections,
        TargetNumTris
```

```
)
{
	FCluster CoarseRepresentation = FindDAGCut(Groups, Clusters, TargetNumTris +4096);

	CoarseRepresentation.Simplify(TargetNumTris);

	TArray< FStaticMeshSection, TInlineAllocator<1> > OldSections = Sections;

	//Need to update the rough representationUcountVto match the new data.
	NumTexCoords = CoarseRepresentation.NumTexCoords;

	//Reconstruct vertex data.
	Verts.Empty(CoarseRepresentation.NumVerts);
	for(uint32 Iter =0, Num = CoarseRepresentation.NumVerts; Iter < Num; ++Iter) {

		FStaticMeshBuildVertex Vertex = {}; Vertex.Position =
		CoarseRepresentation.GetPosition(Iter);                    Vertex.TangentX        =
		FVector::ZeroVector;                FVector::ZeroVector;   Vertex.TangentY        =
		CoarseRepresentation.GetNormal(Iter);                      Vertex.TangentZ        =


		constFVector2D* UVs = CoarseRepresentation.GetUVs(Iter); for(uint32 UVIndex
		=0; UVIndex < NumTexCoords; ++UVIndex) {

			Vertex.UVs[UVIndex] = UVs[UVIndex].ContainsNaN() ? FVector2D::ZeroVector :
UVs[UVIndex];
		}

		if(CoarseRepresentation.bHasColors) {

			Vertex.Color = CoarseRepresentation.GetColor(Iter).ToFColor(false/* sRGB */);
		}

		Verts.Add(Vertex);
	}

	TArray<FMaterialTriangle, TInlineAllocator<128>> CoarseMaterialTris;
	TArray<FMaterialRange, TInlineAllocator<4>> CoarseMaterialRanges;

	//Calculates the material range for the roughness representation.
	BuildMaterialRanges(
		CoarseRepresentation.Indexes,
		CoarseRepresentation.MaterialIndexes,
		CoarseMaterialTris,
		CoarseMaterialRanges);
	check(CoarseMaterialRanges.Num() <= OldSections.Num());

	//reconstructionsectiondata.
	Sections.Reset(CoarseMaterialRanges.Num());
	for(constFStaticMeshSection& OldSection : OldSections) {

		//Add new ones based on calculated material ranges
		section. //Enforce material order withOldSectionsSame.
		constFMaterialRange* FoundRange =
CoarseMaterialRanges.FindByPredicate([&OldSection](constFMaterialRange& Range) {return Range.MaterialIndex ==
OldSection.MaterialIndex; });

		//If their source data does not contain enough triangles, they can actually be removed from the coarse mesh.
```

```
            if(FoundRange)
            {
                //From the original gridsectionCopy attributes.
                FStaticMeshSectionSection(OldSection);


                //RenderSdeycesteicotni.oFnihostrUsenodfe rmake topxPoint and =inde FxooufRaai.nge->RangeStart   *   3;
                Section.NumTriangles            =        FoundRange->RangeLength;
                Section.MinVertexIndex        =  TNumericLimits<uint32>::Max();
                Section.MaxVertexIndex       =   TNumericLimits<uint32>::Min();



                for(uint32 TriangleIndex =0; TriangleIndex < (FoundRange->RangeStart +
FoundRange->RangeLength); ++TriangleIndex)
                {
                    constFMaterialTriangle& Triangle = CoarseMaterialTris[TriangleIndex];

                    //Update the minimum vertex index.
                    Section.MinVertexIndex          =  FMath::Min(Section.MinVertexIndex,
Triangle.Index0);
                    Section.MinVertexIndex          =  FMath::Min(Section.MinVertexIndex,
Triangle.Index1);
                    Section.MinVertexIndex          =  FMath::Min(Section.MinVertexIndex,
Triangle.Index2);

                    //Update the maximum vertex index.
                    Section.MaxVertexIndex          =  FMath::Max(Section.MaxVertexIndex,
Triangle.Index0);
                    Section.MaxVertexIndex          =  FMath::Max(Section.MaxVertexIndex,
Triangle.Index1);
                    Section.MaxVertexIndex          =  FMath::Max(Section.MaxVertexIndex,
Triangle.Index2);
                }

                Sections.Add(Section);
            }
        }
        //Rebuild index data.


        Indexes.Reset();
        for(constFMaterialTriangle& Triangle : CoarseMaterialTris) {

            Indexes.Add(Triangle.Index0);
            Indexes.Add(Triangle.Index1);
            Indexes.Add(Triangle.Index2);
        }

        //Compute tangents.

        CalcTangents(Verts, Indexes);
}
```

6.4.2.7 NaniteEncode

Encode encodes the Nanite resource into Cluster and Cluster group according to FMeshNaniteSettings:

```
// Engine\Source\Developer\NaniteBuilder\Private\NaniteEncode.cpp

void Encode(
	FResources& Resources,
	constFMeshNaniteSettings& TArray<S ettings,
	FCluster >& Clusters, TArray<
	FClusterGroup >& Groups, const
		FBounds & MeshBounds,
	uint32 NumMeshes,
	uint32 NumTexCoords,
	boolbHasColors	)
{
	//Delete degenerate triangles.
	{
		TRACE_CPUPROFILER_EVENT_SCOPE(Nanite::Build::RemoveDegenerateTriangles);
		RemoveDegenerateTriangles(Clusters);
	}

	//Build material scope.
	{
		TRACE_CPUPROFILER_EVENT_SCOPE(Nanite::Build::BuildMaterialRanges);
		BuildMaterialRanges(Clusters);
	}

	//constraintCluster.
#if	USE_CONSTRAINED_CLUSTERS {

		TRACE_CPUPROFILER_EVENT_SCOPE(Nanite::Build::ConstrainClusters);
		ConstrainClusters(Groups, Clusters);
	}
	( . . . . . )
#endif

	//Compute quantized positions.
	{
		TRACE_CPUPROFILER_EVENT_SCOPE(Nanite::Build::CalculateQuantizedPositions);
		//Need toclusterFired after being constrained and split.
		Resources.PositionPrecision = CalculateQuantizedPositionsUniformGrid(Clusters, MeshBounds, Settings);


	}

	//Outputs material range statistics.
	{
		TRACE_CPUPROFILER_EVENT_SCOPE(Nanite::Build::PrintMaterialRangeStats);
		PrintMaterialRangeStats(Clusters);
	}

	TArray<FPage> Pages;
	TArray<FClusterGroupPart> GroupParts;
	TArray<FEncodingInfo> EncodingInfos;

	//Computes the encoded information.
	{
		TRACE_CPUPROFILER_EVENT_SCOPE(Nanite::Build::CalculateEncodingInfos);
		CalculateEncodingInfos(EncodingInfos, Clusters, bHasColors, NumTexCoords);
	}
```

```
        //  distributeClusterarrivePagePage table.
        {
            TRACE_CPUPROFILER_EVENT_SCOPE(Nanite::Build::AssignClustersToPages);
            AssignClustersToPages(Groups, Clusters, EncodingInfos, Pages, GroupParts);
        }

        //BuildClusterThe hierarchy node of the group.
        {
            TRACE_CPUPROFILER_EVENT_SCOPE(Nanite::Build::BuildHierarchyNodes);
            BuildHierarchies(Resources, Groups, GroupParts, NumMeshes);
        }

        //WillClusterandClusterWriting group informationPagePage table. {

            TRACE_CPUPROFILER_EVENT_SCOPE(Nanite::Build::WritePages); WritePages(Resources, Pages,
            Groups, GroupParts, Clusters, EncodingInfos, NumTexCoords);

        }
}
```

The above coding process involves many important interfaces. Let's analyze them one by one:

```
// Engine\Source\Developer\NaniteBuilder\Private\NaniteEncode.cpp

static void RemoveDegenerateTriangles(TArray<FCluster>& Clusters) {

    //Deleting in parallelClusterDegenerate
    triangles of lists. ParallelFor( Clusters.Num(),
        [&]( uint32 ClusterIndex ) {

            RemoveDegenerateTriangles(Clusters[ClusterIndex]);
        } );
}

//Delete a singleClusterThe degenerate triangle of .
static void RemoveDegenerateTriangles(FCluster& Cluster) {

    uint32 NumOldTriangles = Cluster.NumTris; uint32
    NumNewTriangles =0;

    for(uint32 OldTriangleIndex =0; OldTriangleIndex < NumOldTriangles; OldTriangleIndex++)

    {
        uint32 i0 = Cluster.Indexes[OldTriangleIndex *3+0]; uint32 i1 =
        Cluster.Indexes[OldTriangleIndex *3+1]; uint32 i2 =
        Cluster.Indexes[OldTriangleIndex *3+2]; uint32 mi =
        Cluster.MaterialIndexes[OldTriangleIndex];

        //  If it is not a degenerate triangle, then3The data of the vertices must be different from each other. Author's note: Maybe an optimization can be made here,
        //  such as the distance between any two vertices of a triangle is less than a certain threshold.                                    (0.01f)Degeneration
Triangle.
        if(i0 != i1 && i0 != i2 && i1 != i2) {

            Cluster.Indexes[NumNewTriangles   *3+0]   =   i0;
            Cluster.Indexes[NumNewTriangles   *3+1]   =   i1;
            Cluster.Indexes[NumNewTriangles   *3+2]   =   i2;
            Cluster.MaterialIndexes[NumNewTriangles] = mi;
```

```
                NumNewTriangles++;
            }
        }
        Cluster . N um T ris                    =       N um N ew T riangles ;
        C luster . I ndexes . S et N um ( N um N ew T riangles C luster . Material I ndexes . S et N um ( N um N ew T ri    *       3
                                                                                                                              a   r

}

//WillClusterTriangle classification into material ra,nge Add material range toCluster.
static void BuildMaterialRanges(TArray<FCluster>& Clusters) {

    //Parallel processing.
    ParallelFor( Clusters.Num(),
        [&]( uint32 ClusterIndex ) {

            BuildMaterialRanges(Clusters[ClusterIndex]);
        } );
}

static void BuildMaterialRanges(FCluster& Cluster) {

    TArray<FMaterialTriangle, TInlineAllocator<128>>        MaterialTris;
    //Build a singleClusterRange of
    materials. BuildMaterialRanges(

    Cluster.Indexes,
    Cluster.MaterialIndexes,
    MaterialTris,
    Cluster.MaterialRanges);


    //Write the index back toCluster.
    for(uint32 Triangle =0; Triangle < Cluster.NumTris; ++Triangle) {

        Cluster.Indexes[Triangle       *    3+       0]  =  MaterialTris[Triangle].Index0;
        Cluster.Indexes[Triangle       *    3+       1]  =  MaterialTris[Triangle].Index1;
        Cluster.Indexes[Triangle       *    3+       2] = MaterialTris[Triangle].Index2;
        Cluster.MaterialIndexes[Triangle] = MaterialTris[Triangle].MaterialIndex;
    }
}

//constraintCluster.
static void ConstrainClusters(TArray<FClusterGroup>& ClusterGroups, TArray<FCluster>& Clusters)

{
    //Compute statistics.
    uint32 TotalOldTriangles =0; uint32
    TotalOldVertices =0;
    for(constFCluster& Cluster : Clusters ) {

        TotalOldTriangles       + = Cluster.NumTris;
        TotalOldVertices       += Cluster.NumVerts;
    }

    //Parallel ConstraintsCluster,Determine whether to use banded index.
    ParallelFor( Clusters.Num(),
        [&]( uint32 i )
        {
```

```
#ifUSE_STRIP_INDICES//Use a banded index.
                    FStripifier Stripifier;
                    Stripifier.ConstrainAndStripifyCluster(Clusters[i]); No striped indexing
#else//       is used.

                    ConstrainClusterFIFO(Clusters[i]);
#endif
            }   ) ;

        uint32 TotalNewTriangles =0; uint32
        TotalNewVertices =0;

        //constraintcluster.
        constuint32 NumOldClusters = Clusters.Num(); for( uint32 i =
        0; i < NumOldClusters; i++ ) {

            TotalNewTriangles += Clusters[ i ].NumTris;
            TotalNewVertices += Clusters[ i ].NumVerts;

            //    ifClusterToo many vertices (more than256), split them.
            if( Clusters[ i ].NumVerts >256) {

                FCluster ClusterA, ClusterB;

                uint32 NumTrianglesA = Clusters[ i ].NumTris /2;
                uint32 NumTrianglesB = Clusters[ i ].NumTris - NumTrianglesA;

                BuildClusterFromClusterTriangleRange( Clusters[ i ], ClusterA,0,
NumTrianglesA );
                BuildClusterFromClusterTriangleRange( Clusters[ i ], ClusterB, NumTrianglesA,
NumTrianglesB );

                Clusters[i] = ClusterA;
                ClusterGroups[ ClusterB.GroupIndex ].Children.Add( Clusters.Num() );
                Clusters.Add( ClusterB );
            }
        }
        //Compute statistics.
        uint32 TotalNewTrianglesWithSplits =0; uint32
        TotalNewVerticesWithSplits =0; for(constFCluster&
        Cluster : Clusters ) {



            TotalNewTrianglesWithSplits              + = Cluster.NumTris;
            TotalNewVerticesWithSplits             += Cluster.NumVerts;
        }

        (.....)
}

//Computes a uniform grid of quantized positions.

staticint32CalculateQuantizedPositionsUniformGrid(TArray< FCluster >& Clusters,const FBounds & MeshBounds,const
FMeshNaniteSettings& Settings) {

        //forEASimplify global quantization values.
        constint32 MaxPositionQuantizedValue                                    = (1<< MAX_POSITION_QUANTIZATION_BITS) -1;

        int32 PositionPrecision = Settings.PositionPrecision; if(PositionPrecision
        == MIN_int32)
```

```
{
    //Automatic: Guess the required precision from the leaf-level bounds.
    const floatMaxSize = MeshBounds.GetExtent().GetMax();

    // ///// Heuristic: If the grid is denser, a higher resolution is needed.
    doubleTotalLogSize =0.0; int32 TotalNum =0; for(constFCluster& Cluster : Clusters) {
        Usageclusterthe geometric mean of the sizes is used as a proxy for density. Another interpretation: bit precision isclusterthe desired average value. For approximately the same sizecluster, which gives results very similar to the old quantization code.

        if(Cluster.MipLevel ==0) {
            floatExtentSize = Cluster.Bounds.GetExtent().Size(); if(ExtentSize >0.0) {
                TotalLogSize += FMath::Log2(ExtentSize); TotalNum++;
            }
        }
    }
    double A vg Log Size = Total Number > 0 ? To a l Position Precision = 7 - FM ath : : R ound T o l n

    //Truncate precision. Users now need to explicitly select a minimum precision setting.
    //These settings may cause problems and contribute little to disk size savings (approximately0.4%),So it should not be automatically selected Select them.
    //For example, a very low-resolution road or building frame may not require much precision in isolation. But in one scene, Considerably higher precision is required as smaller grids are placed above or inside.
    constint32 AUTO_MIN_PRECISION =4; //The minimum precision is1/16cm.
    PositionPrecision = FMath::Max(PositionPrecision, AUTO_MIN_PRECISION);
}

//Calculate the quantization ratio.
floatQuantizationScale = FMath::Exp2((float)PositionPrecision);

//Ensure allclusterare all encodable. A large enoughclusterMay reach21bpcIf it does, scale it down until it's just right.
for(constFCluster& Cluster : Clusters) {

    constFBounds& Bounds = Cluster.Bounds;

    int32 Iterations =0; while(
    true)
    {
        floatMinX = FMath::RoundToFloat(Bounds.Min.X * QuantizationScale); floatMinY = FMath::RoundToFloat(Bounds.Min.Y * QuantizationScale); floatMinZ = FMath::RoundToFloat(Bounds.Min.Z * QuantizationScale);

        floatMaxX = FMath::RoundToFloat(Bounds.Max.X * QuantizationScale); floatMaxY = FMath::RoundToFloat(Bounds.Max.Y * QuantizationScale); floatMaxZ = FMath::RoundToFloat(Bounds.Max.Z * QuantizationScale);

        if(MinX >= (double)MIN_int32 && MinY >= (double)MIN_int32 && MinZ >= (double)MIN_int32 && // MIN_int32/MAX_int32 is not representable in float
            MaxX <= (double)MAX_int32 && MaxY <= (double)MAX_int32 && MaxZ <= (double)MAX_int32 &&
```

```
                ((int32)MaxX - (int32)MinX) <= MaxPositionQuantizedValue && ((int32)MaxY -
(int32)MinY) <= MaxPositionQuantizedValue && ((int32)MaxZ - (int32)MinZ) <=
MaxPositionQuantizedValue)
            {
                break;
            }

            QuantizationScale *=        0.5f;
            PositionPrecision--;
            check(++Iterations <        100);        // Endless loop?
        }
    }

    const floatRcpQuantizationScale =1.0f/QuantizationScale;

    //Process position quantization in parallel.
    ParallelFor(Clusters.Num(), [&](uint32 ClusterIndex) {

        FCluster& Cluster = Clusters[ClusterIndex];

        constuint32 NumClusterVerts = Cluster.NumVerts; constuint32
        ClusterShift = Cluster.QuantizedPosShift;

        Cluster.QuantizedPositions.SetNumUninitialized(NumClusterVerts);

        //Quantification FBit InPlatcVe.ector  IntClusterMax          =      {
        MIN_int32, FIntVector IntClusterMin = { MAX_int32,          MIN_int32,  MIN_int32  };
                                                                    MAX_int32, MAX_int32 };

        for(uint32 i =0; i < NumClusterVerts; i++) {

            constFVector Position = Cluster.GetPosition(i);

            FIntVector& IntPosition = Cluster.QuantizedPositions[i];float PosX =
            FMath::RoundToFloat(Position.X * QuantizationScale);float PosY =
            FMath::RoundToFloat(Position.Y * QuantizationScale);float PosZ =
            FMath::RoundToFloat(Position.Z * QuantizationScale);

            IntPosition = FIntVector((int32)PosX, (int32)PosY, (int32)PosZ);

            IntClusterMax.X = FMath::Max(IntClusterMax. .Z); IntClusterMin.X =
            FMath::Min(IntClusterMin.X,     IntPosition.X);     IntClusterMin.Y     =
            FMath::Min(IntClusterMin.Y,     IntPosition.Y);     IntClusterMin.Z     =
            FMath::Min(IntClusterMin .Z, IntPosition.Z);



        }

        //liveco Storanges.32 NumBitsX = FMath::CeilLogTwo(IntClusterMax.X - IntClusterMin.X +
        1);constuint32 NumBitsY = FMath::CeilLogTwo(IntClusterMax.Y - IntClusterMin.Y + 1);constuint32 NumBitsZ =
        FMath::CeilLogTwo(IntClusterMax.Z - IntClusterMin.Z + 1); check(NumBitsX <=
        MAX_POSITION_QUANTIZATION_BITS); check(NumBitsY <= MAX_POSITION_QUANTIZATION_BITS);
                                                                    check(NumBitsZ                  <=
        MAX_POSITION_QUANTIZATION_BITS);


        for(uint32 i =0; i < NumClusterVerts; i++) {
```

```
                FIntVector& IntPosition = Cluster.QuantizedPositions[i];

                //Update floating point positions with quantized data.
                Cluster.GetPosition(i) = FVector(IntPosition.X * RcpQuantizationScale,
IntPosition.Y * RcpQuantizationScale, IntPosition.Z * RcpQuantizationScale);

                IntPosition.X -= IntClusterMin.X; IntPosition.Y -= IntClusterMin.Y; IntPosition.Z -= IntClusterMin.Z;
                check(IntPosition.X >=0&& IntPosition.X < (1<< NumBitsX)); check(IntPosition.Y >=0 && IntPosition.Y < (1<< NumBitsY)); check(IntPosition.Z >=0&& IntPosition.Z < (1<< NumBitsZ));

            }

            //Update the bounding box.
            Cluster.Bounds.Min = FVector(IntClusterMin.X * RcpQuantizationScale, IntClusterMin.Y *
RcpQuantizationScale, IntClusterMin.Z * RcpQuantizationScale);
            Cluster.Bounds.Max = FVector(IntClusterMax.X * RcpQuantizationScale, IntClusterMax.Y *
RcpQuantizationScale, IntClusterMax.Z * RcpQuantizationScale);

            Cluster.MeshBoundsMin = FVector::ZeroVector;
            Cluster.MeshBoundsDelta = FVector(RcpQuantizationScale);

            Cluster.QuantizedPosBits = FIntVector(NumBitsX, NumBitsY, NumBitsZ);
            Cluster.QuantizedPosStart = IntClusterMin;
            Cluster.QuantizedPosShift = 0;

        } );

        returnPositionPrecision;
}

//Calculate a groupClusterEncoding information.
static void CalculateEncodingInfos(TArray<FEncodingInfo>& EncodingInfos,const
TArray<Nanite::FCluster>& Clusters,boolbHasColors, uint32 NumTexCoords) {

        uint32 NumClusters = Clusters.Num();
        EncodingInfos.SetNumUninitialized(NumClusters);

        for(uint32 i =0; i < NumClusters; i++) {

            CalculateEncodingInfo(EncodingInfos[i], Clusters[i], bHasColors, NumTexCoords);
        }
}

//Calculate a singleClusterEncoding information.
static void CalculateEncodingInfo(FEncodingInfo& Info,constNanite::FCluster& Cluster, boolbHasColors, uint32
NumTexCoords) {

        constuint32 NumClusterVerts = Cluster.NumVerts; constuint32
        NumClusterTris = Cluster.NumTris;

        FMemory::Memzero(Info);

        //Writes a triangle index. The index is stored in a dense bit stream, and each index usesceil(log2(NumClusterVerices))The shader implements unaligned bitstream reading to support this.
        constuint32 BitsPerIndex = NumClusterVerts >1?
(FGenericPlatformMath::FloorLog2(NumClusterVerts -1) +1) :0;
```

```
        constuint32 BitsPerTriangle = BitsPerIndex +2*5; offsets                    // Base index + two 5-bit

        Info.BitsPerIndex = BitsPerIndex;

        //Calculate page information.

        FPageSections& GpuSizes = Info.GpuSizes;
        GpuSizes.Cluster =sizeof(FPackedCluster);
        GpuSizes.MaterialTable = CalcMaterialTableSize(Cluster) *sizeof(uint32); GpuSizes.DecodeInfo =
        NumTexCoords *sizeof(FUVRange); GpuSizes.Index = (NumClusterTris * BitsPerTriangle +31) /32*4;


#ifUSE_UNCOMPRESSED_VERTEX_DATA//Use uncompressed vertex data.
        constuint32 AttribBytesPerVertex = (3*sizeof(float) +sizeof(uint32) + NumTexCoords
*  2  *  sizeof(float));

        Info.BitsPerAttribute = AttribBytesPerVertex *8; Info.ColorMin =
        FIntVector4(0,0,0,0); Info.ColorBits = FIntVector4(8,8,8,8
        Info.ColorMode );
                                =  VERTEX_COLOR_MODE_VARIABLE;
        Info.UVPrec =         0;

        GpuSizes.Position = NumClusterVerts *3*sizeof(float);
        GpuSizes.Attribute = NumClusterVerts * AttribBytesPerVertex;
#else//Use compressed vertex data.
        Info.BitsPerAttribute =2* NORMAL_QUANTIZATION_BITS;

        check(NumClusterVerts >0);
        const boolbIsLeaf = (Cluster.GeneratingGroupIndex == INVALID_GROUP_INDEX);
        //Vertex color.

        Info.ColorMode = VERTEX_COLOR_MODE_WHITE; Info.ColorMin =
        FIntVector4(255,255,255,255); if(bHasColors)

        {
                FIntVector4 ColorMin = FIntVector4(255,255,255,255); FIntVector4 ColorMax
                = FIntVector4(0,0,0,0); for(uint32 i =0; i < NumClusterVerts; i++) {


                        FColor Color = Cluster.GetColor(i).ToFColor(false); ColorMin.X =
                        FMath::Min(ColorMin.X, (int32)Color.R); ColorMin.Y =
                        FMath::Min(ColorMin.Y, (int32)Color.G); ColorMin.Z =
                        FMath: :Min(ColorMin.Z, (int32)Color.B); ColorMin.W =
                        FMath::Min(ColorMin.W, (int32)Color.A); ColorMax.X =
                        FMath::Max(ColorMax.X, ( int32)Color.R); ColorMax.Y =
                        FMath::Max(ColorMax.Y, (int32)Color.G); ColorMax.Z =
                        FMath::Max(ColorMax.Z, (int32)Color.B); ColorMax .W =
                        FMath::Max(ColorMax.W, (int32)Color.A);

                }

                constFIntVector4 ColorDelta = ColorMax - ColorMin;const int32 R_Bits =
                FMath::CeilLogTwo(ColorDelta.X +1);const int32 G_Bits =
                FMath::CeilLogTwo(ColorDelta.Y +1);const int32 B_Bits =
                FMath::CeilLogTwo(ColorDelta.Z +1);const int32 A_Bits =
                FMath::CeilLogTwo(ColorDelta.W +1);

                uint32 NumColorBits = R_Bits + G_Bits + B_Bits + A_Bits; Info.BitsPerAttribute
                += NumColorBits;
                Info.ColorMin = ColorMin;
```

```
        Info.ColorBits = FIntVector4(R_Bits, G_Bits, B_Bits, A_Bits); if(NumColorBits >0) {


            Info.ColorMode = VERTEX_COLOR_MODE_VARIABLE;
        }
        else
        {
            if(ColorMin.X ==255&& ColorMin.Y ==255&& ColorMin.Z ==255&& ColorMin.W
== 255)
                Info.ColorMode      =  VERTEX_COLOR_MODE_WHITE;
            else
                Info.ColorMode      =  VERTEX_COLOR_MODE_CONSTANT;
        }
    }

    for( uint32 UVIndex =0; UVIndex < NumTexCoords; UVIndex++ ) {

        FGeometryEncodingUVInfo& UVInfo = Info.UVInfos[UVIndex]; //
            Block-compressed texture coordinates.
        //  Texture coordinates relative toClusterMinimum/MaximumUVCoordinate storage.
        //  UVThe seams produce very large sparse bounding rectangles. To mitigate this, the largest gap is betweenUEdge VThe bounding rectangle is excluded from the encoding space
                                                                                                                                                    .
        //Decoding this is pretty easy:UV += (UV >= GapStart) ? GapRange : 0; //
        Generate sequenceaof
              TArrayeUaffnudmoVbaeGrrotu>p.UValues; TArray<float>
        VValues;
        UValues.AddUninitialized(NumClusterVerts);
        VValues.AddUninitialized(NumClusterVerts); for(uint32 i =0; i
        < NumClusterVerts; i++) {


            constFVector2D& UV = Cluster.GetUVs(i)[ UVIndex ]; UValues[i]
                        =  UV.X;
            VValues[i]      = UV.Y;
        }

        UValues.Sort();
        VValues.Sort();

        //Find the orderuOf the largest gap between
        FVector2D LargestGapStart = FVector2D(UValues[0], VValues[0]); FVector2D
        LargestGapEnd = FVector2D(UValues[0], VValues[0]); for(uint32 i =0; i <
        NumClusterVerts -1; i++) {


            if(UValues[i +1] - UValues[i] > LargestGapEnd.X - LargestGapStart.X) {

                LargestGapStart.X = UValues[i];
                LargestGapEnd.X = UValues[i +1];
            }
            if          ( VV alues [ i                          +       1]          -       VV alues [ i ]                          >       L       rg
            {
                LargestGapStart.Y = VValues[i];
                LargestGapEnd.Y = VValues[i +1];
            }
        }

        constFVector2D UVMin = FVector2D(UValues[0], VValues[0]); constFVector2D UVMax =
        FVector2D(UValues[NumClusterVerts -1], VValues[NumClusterVerts -1]);

        constFVector2D UVDelta = UVMax - UVMin;
```

```
            constFVector2D UVRcpDelta = FVector2D( UVDelta.X          UVDelta.X > SMALL_NUMBER ?1.0f/
               :   0.0f,
                                                                      UVDelta.Y > SMALL_NUMBER ?1.0f/
UVDelta.Y      :   0.0f);

            constFVector2D NonGapLength = FVector2D::Max(UVDelta - (LargestGapEnd - LargestGapStart),
FVector2D(0.0f,0.0f));
            constFVector2D NormalizedGapStart = (LargestGapStart - UVMin) * UVRcpDelta; constFVector2D
            NormalizedGapEnd = (LargestGapEnd - UVMin) * UVRcpDelta;

            constFVector2D NormalizedNonGapLength = NonGapLength * UVRcpDelta;

#if1
            const floatTexCoordUnitPrecision = (1<<14); mode that                    //TODO:Implement UI + 'Auto'
decides when this is necessary.

            int32 TexCoordBitsU =0; if
            (UVDelta.X >0) {

                  //Even whenNonGapLength=0hour,UVDeltais non-zero, so at least2Values(1bit)To distinguish high from low.
                  int32 NumValues = FMath::Max(FMath::CeilToInt(NonGapLength.X *
TexCoordUnitPrecision),2);
                  //Restricted to12Position, from the temporaryhackIt is good enough to know.
                  TexCoordBitsU = FMath::Min((int32)FMath::CeilLogTwo(NumValues),12);
            }

            int32 TexCoordBitsV =0; if
            (UVDelta.Y >0) {

                  int32 NumValues = FMath::Max(FMath::CeilToInt(NonGapLength.Y *
TexCoordUnitPrecision),2);
                  TexCoordBitsV = FMath::Min((int32)FMath::CeilLogTwo(NumValues),12);
            }
# else
            //temporaryhackTo fix encoding issues.
            constint32 TexCoordBitsU =12; constint32
            TexCoordBitsV =12;
#endif

            //deal withUsVMark and size.
            Info.UVPrec |= ((TexCoordBitsV <<4) | TexCoordBitsU) << (UVIndex *8);

            constint32 TexCoordMaxValueU = (1<< TexCoordBitsU) -1; constint32
            TexCoordMaxValueV = (1<< TexCoordBitsV) -1;

            constint32 NU = (int32)FMath::Clamp(NormalizedNonGapLength.X > SMALL_NUMBER ?
(TexCoordMaxValueU -2)/NormalizedNonGapLength.X:0.0f, (float)TexCoordMaxValueU, (float)0xFFFF);

            constint32 NV = (int32)FMath::Clamp(NormalizedNonGapLength.Y > SMALL_NUMBER ?
(TexCoordMaxValueV -2) / NormalizedNonGapLength.Y :0.0f, (float)TexCoordMaxValueV, (float)0xFFFF);


            int32 GapStartU = TexCoordMaxValueU +1; int32
            GapStartV = TexCoordMaxValueV +1; int32 GapLengthU
            =0; int32 GapLengthV = 0;if(NU > TexCoordMaxValueU)
```

```
            {
                GapStartU = int32(NormalizedGapStart.X * NU +0.5f) +1; constint32 GapEndU =
                int32(NormalizedGapEnd.X * NU +0.5f); GapLengthU = FMath::Max(GapEndU -
                GapStartU,0);
            }
            if          ( N.V.          >       T ex C oord Max Value V )
            {
                GapStartV = int32(NormalizedGapStart.Y * NV +0.5f) +1; constint32 GapEndV =
                int32(NormalizedGapEnd.Y * NV +0.5f); GapLengthV = FMath::Max(GapEndV -
                GapStartV,0);
            }

            UVInfo.UVRange.Min = UVMin;
            UVInfo.UVRange.Scale = FVector2D(NU >0? UVDelta.X / NU :0.0f, NV >0? UVDelta.Y / NV :0.0f);


            check(GapStartU >=0); check(GapStartV >= 0);
                        UVInfo.UVRange.GapStart[0]                  =
                        GapSUtaVrItnUfo;.UVRange.GapStart[1     =
            GapStartV;      UVInfo.UVRange.GapLength[0          =
            GapLengthU;     UVInfo.UVRange.GapLength[1]         =
            GapLengthV;

            UVInfo.UVDelta          =       UVDelta;
            UVInfo.UVRcpDelta       = UVRcpDelta;
            UVInfo.NU = NU; UVInfo.NV = NV;


            Info.BitsPerAttribute += TexCoordBitsU + TexCoordBitsV;
        }

        constuint32 PositionBitsPerVertex = Cluster.QuantizedPosBits.X +
Cluster.QuantizedPosBits.Y + Cluster.QuantizedPosBits.Z;
        GpuSizes.Position = (NumClusterVerts * PositionBitsPerVertex +31) /32*4; GpuSizes.Attribute =
        (NumClusterVerts * Info.BitsPerAttribute +31) /32*4;
#endif
}

/*
Build StreamPage
Pagelayout:
        Fixup Chunk     (Only loaded intoCPUMemory)
        FPackedCluster
        MaterialRangeTable
        GeometryData

* /                  static                      void
AssignClustersToPages(
        TArray< FClusterGroup >& ClusterGroups, TArray<
        FCluster >& Clusters,
        constTArray< FEncodingInfo >& EncodingInfos,
        TArray<FPage>& Pages,
        TArray<FClusterGroupPart>& )        Parts

{
        check(Pages.Num()       == 0);
        check(Parts.Num()       == 0);

        constuint32 NumClusterGroups = ClusterGroups.Num();
```

```
        Pages.AddDefaulted();

        SortGroupClusters(ClusterGroups, Clusters); TArray<uint32>
        ClusterGroupPermutation =
        CalculateClusterGroupPermutation(ClusterGroups);

        for(uint32 i =0; i < NumClusterGroups; i++) {

            //Pick the best nextGroup.
            uint32 GroupIndex = ClusterGroupPermutation[i]; FClusterGroup&
            Group = ClusterGroups[GroupIndex]; uint32 GroupStartPage =
            INVALID_PAGE_INDEX;


            for(uint32 ClusterIndex : Group.Children) {

                //Pick the best nextCluster.
                FCluster& Cluster = Clusters[ClusterIndex];
                constFEncodingInfo& EncodingInfo = EncodingInfos[ClusterIndex];


                //JoinPage.
                FPage* Page = &Pages.Top();
                if(Page->GpuSizes.GetTotal() + EncodingInfo.GpuSizes.GetTotal() >
CLUSTER_PAGE_GPU_SIZE || Page->NumClusters +1> MAX_CLUSTERS_PER_PAGE)
                {
                    // PageFull, need to add one.
                    Pages.AddDefaulted();
                    Page = &Pages.Top();

                }

                //Check whether newFClusterGroupPart.
                if(Page->PartsNum ==0|| Parts[Page->PartsStartIndex + Page->PartsNum -
1].GroupIndex      != GroupIndex)
                {
                    if(Page->PartsNum ==0) {

                        Page->PartsStartIndex = Parts.Num();
                    }
                    P age - > P arts N um + + ;

                    FClusterGroupPart& Part = Parts.AddDefaulted_GetRef(); Part.GroupIndex
                    = GroupIndex;
                }

                //Adud intotc3l2u sPtaegraerInridveepx a=g Pea.ges.Num() -
                1; uint32 PartIndex = Parts.Num() - 1;


                FClusterGroupPart& Part = Parts.Last(); if
                (Part.Clusters.Num() ==0) {

                    Part.PageClusterOffset = Page->NumClusters;
                    Part.PageIndex = PageIndex;
                }
                P art . C lusters . A dd ( C luster Index ) ;
                check ( Part . C lusters . N um ( ) < = MAX _ CLU                                ST

                Cluster.GroupPartIndex = PartIndex;
```

```
            if(GroupStartPage == INVALID_PAGE_INDEX) {

                GroupStartPage = PageIndex;
            }

            Page->GpuSizes += EncodingInfo.GpuSizes; Page-
            >NumClusters++;
        }

        Group.PageIndexStart = GroupStartPage; Group.PageIndexNum =
        Pages.Num() - GroupStartPage; check(Group.PageIndexNum
                                            > =1);
        check(Group.PageIndexNum        <=   MAX_GROUP_PARTS_MASK);
    }

    //Recalculategroup partThe bounding box of .
    for(FClusterGroupPart& Part : Parts) {

        check(Part.Clusters.Num() <= MAX_CLUSTERS_PER_GROUP);
        check(Part.PageIndex < (uint32)Pages.Num());

        FBounds Bounds;
        for(uint32 ClusterIndex : Part.Clusters) {

            Bounds += Clusters[ClusterIndex].Bounds;
        }
        Part . Bounds                       =       Bounds ;
    }
}

//BuildClusterGroupHierarchical structure.
static void BuildHierarchies(FResources& Resources,constTArray<FClusterGroup>& Groups,
TArray<FClusterGroupPart>& Parts, uint32 NumMeshes)
{
    TArray<TArray<uint32>> PartsByMesh;
    PartsByMesh.SetNum(NumMeshes);

    //Willgroup partAssigned to the grid they belong to.
    constuint32 NumTotalParts = Parts.Num();
    for(uint32 PartIndex =0; PartIndex < NumTotalParts; PartIndex++) {

        FClusterGroupPart& Part = Parts[PartIndex];
        PartsByMesh[Groups[Part.GroupIndex].MeshIndex].Add(PartIndex);
    }

    for(uint32 MeshIndex =0; MeshIndex < NumMeshes; MeshIndex++) {

        constTArray<uint32>& PartIndices = PartsByMesh[MeshIndex]; constuint32
        NumParts = PartIndices.Num();

        int32 MaxMipLevel =0;
        for(uint32 i =0; i < NumParts; i++) {

            MaxMipLevel = FMath::Max(MaxMipLevel,
Groups[Parts[PartIndices[i]].GroupIndex].MipLevel);
        }

        TArray< FIntermediateNode >             Nodes;
```

```
Nodes.SetNum(NumParts);

//For each gridLODThe hierarchy constructs leaf nodes.
TArray<TArray<uint32>> NodesByMip;
NodesByMip.SetNum(MaxMipLevel +1); for(uint32
i =0; i < NumParts; i++) {

    constuint32 PartIndex = PartIndices[i]; constFClusterGroupPart& Part =
    Parts[PartIndex]; constFClusterGroup& Group =
    Groups[Part.GroupIndex];

    constint32 MipLevel = Group.MipLevel;
    FIntermediateNode& Node = Nodes[i]; Node.Bound
    = Part.Bounds;
    Node.PartIndex      = PartIndex;
    Node.MipLevel      = Group.MipLevel;
    Node.bLeaf =      true;
    NodesByMip[Group.MipLevel].Add(i);
}

uint32 RootIndex =0; if
(Nodes.Num() ==1) {

    //It is just a leaf node and requires special setup because the root node is always
    an internal node. FIntermediateNode& Node = Nodes.AddDefaulted_GetRef();
    Node.Children.Add(0); Node.Bound
                =  Nodes0].Bound;
    RootIndex    =  1;

}
else
{
    //  Build a hierarchy (Hierarchy):
    //  NaniteThe grid contains many  LODLevelCluster data.  Different levelsCluster  The size can vary greatly, which is helpful for building a good
OKHierarchyIt seems like a challenge.
    //  In addition to the visibility bounding box, theHierarchyIt also tracks the conservativeness of child nodes.LODError metric. As long as the child
    //  nodes are visible and conservativeLODThe error is no more detailed than what we are looking for, and the runtime traversal will drop.LOD
    //                          ofCluster  We must be very careful when doing this, because the less detailedClusterIt is easy to cause the bounding box and
Inflation of error metrics.
    //We have tried manyLODA mixed approach, but currently it seems that for eachLODLevel build separateHierarchyThen build
These Hierarchy  of  Hierarchy, you can get the best and most predictable results.
    TArray<uint32>      LevelRoots;
    for(int32 MipLevel =0; MipLevel <= MaxMipLevel; MipLevel++) {

        if(NodesByMip[MipLevel].Num() >0) {

            //formipHierarchical constructionhierarchy,A top-down separation method was used.
            uint32 NodeIndex = BuildHierarchyTopDown(Nodes, NodesByMip[MipLevel],
true);

            if(Nodes[NodeIndex].bLeaf || Nodes[NodeIndex].Children.Num() ==
MAX_BVH_NODE_FANOUT)
            {
                //Leaf or filler nodes, added directly.
                LevelRoots.Add(NodeIndex);

            }
            else
            {
                //Incomplete node. Discard the encoding and add the child node as the root node.
```

```
                    LevelRoots.Append(Nodes[NodeIndex].Children);
                }
            }
        }
        // Building the top layerhierarchy , yesMIP Hierarchies of hi
        Root Index = B uild H ierarchy T op D own ( N o d e
    }

    check(Nodes.Num() >0);

#ifBVH_BUILD_WRITE_GRAPHVIZ
  WriteDotGraph(Nodes);
#endif

    TArray< FHierarchyNode > HierarchyNodes; BuildHierarchyRecursive(HierarchyNodes, Nodes,
    Groups, Parts, RootIndex);

    //ConversionhierarchyInto compressed format.

    constuint32 NumHierarchyNodes = HierarchyNodes.Num(); constuint32
    PackedBaseIndex = Resources.HierarchyNodes.Num();
    Resources.HierarchyRootOffsets.Add(PackedBaseIndex);
    Resources.HierarchyNodes.AddDefaulted(NumHierarchyNodes); for(uint32 i =0; i <
    NumHierarchyNodes; i++) {

        //compressionHierarchynode.
        PackHierarchyNode(Resources.HierarchyNodes[PackedBaseIndex + i],
HierarchyNodes[i], Groups, Parts);
    }
  }
}

//Write to the page table.

static void WritePages( FResources& Resources,
                        TArray<FPage>& Pages,
                        const TArray<FClusterGroup>& Groups,
                        const TArray<FClusterGroupPart>& TArray<PFaCrtlus,ster>&
                        const Clusters, TArray<FEncodingInfo>& EncodingInfos,
                        const NumTexCoords)
                        uint32
{
  check(Resources.PageStreamingStates.Num() ==0);

  const boolbLZCompress =true;

  TArray< uint8 > StreamableBulkData;

  constuint32 NumPages = Pages.Num(); constuint32
  NumClusters = Clusters.Num();
  Resources.PageStreamingStates.SetNum(NumPages);

  //deal withFixupChunk.
  uint32 TotalGPUSize =0;
  TArray<FFixupChunk> FixupChunks;
  FixupChunks.SetNum(NumPages);
  for(uint32 PageIndex =0; PageIndex < NumPages; PageIndex++) {

      constFPage& Page = Pages[PageIndex]; FFixupChunk&
      FixupChunk = FixupChunks[PageIndex];
```

```
            FixupChunk.Header.NumClusters = Page.NumClusters;

            uint32 NumHierarchyFixups =0;
            for(uint32 i =0; i < Page.PartsNum; i++) {

                  constFClusterGroupPart& Part = Parts[Page.PartsStartIndex + i]; NumHierarchyFixups
                  += Groups[Part.GroupIndex].PageIndexNum;
            }

            FixupChunk.Header.NumHierachyFixups = NumHierarchyFixups; must be set        // NumHierarchyFixups
before writing cluster fixups
            TotalGPUSize += Page.GpuSizes.GetTotal();
      }

      //TowardsPageAdded additional corrections.
      for(constFClusterGroupPart& Part : Parts) {

            check(Part.PageIndex < NumPages);

            constFClusterGroup& Group = Groups[Part.GroupIndex]; for(uint32
            ClusterPositionInPart =0; ClusterPositionInPart < (uint32)Part.Clusters.Num();
ClusterPositionInPart++)
            {
                  constFCluster& Cluster = Clusters[Part.Clusters[ClusterPositionInPart]]; if
                  (Cluster.GeneratingGroupIndex != INVALID_GROUP_INDEX) {

                        constFClusterGroup& GeneratingGroup =
Groups[Cluster.GeneratingGroupIndex];
                        check(GeneratingGroup.PageIndexNum >=1);

                        if(GeneratingGroup.PageIndexStart == Part.PageIndex &&
GeneratingGroup.PageIndexNum ==          1)
                              continue;          // Dependencies already met by current page. Fixup
directly instead.

                        uint32 PageDependencyStart = GeneratingGroup.PageIndexStart; uint32
                        PageDependencyNum = GeneratingGroup.PageIndexNum;
                        RemoveRootPagesFromRange(PageDependencyStart, PageDependencyNum);                  //
Root page should never be a dependency

                        constFClusterFixup ClusterFixup = FClusterFixup(Part.PageIndex,
Part.PageClusterOffset + ClusterPositionInPart, PageDependencyStart, PageDependencyNum);
                        for(uint32 i =0; i < GeneratingGroup.PageIndexNum; i++) {

                              FFixupChunk& FixupChunk = FixupChunks[GeneratingGroup.PageIndexStart +
i];
                              FixupChunk.GetClusterFixup(FixupChunk.Header.NumClusterFixups++) =
ClusterFixup;
                        }
                  }
            }
      }
      //Generatepagerely.

      for(uint32 PageIndex =0; PageIndex < NumPages; PageIndex++) {

            constFFixupChunk& FixupChunk = FixupChunks[PageIndex];
            FPageStreamingState& PageStreamingState =
```

```
Resources.PageStreamingStates[PageIndex];
            PageStreamingState.DependenciesStart = Resources.PageDependencies.Num();

            for(uint32 i =0; i < FixupChunk.Header.NumClusterFixups; i++) {

                  uint32 FixupPageIndex = FixupChunk.GetClusterFixup(i).GetPageIndex();
                  check(FixupPageIndex < NumPages);
                  if(IsRootPage(FixupPageIndex) || FixupPageIndex == PageIndex)                    // Never
emit dependencies to ourselves or a root page.
                        continue;

                  //    It is added only if it is not in the collection.
                  //    O(n^2),But in reality the number of dependencies will be smaller.
                  boolbFound =false;
                  for(uint32 j = PageStreamingState.DependenciesStart; j <
(uint32)Resources.PageDependencies.Num(); j++)
                  {
                        if(Resources.PageDependencies[j] == FixupPageIndex) {

                              bFound =true;
                              break;
                        }
                  }

                  if(bFound)
                        continue;

                  Resources.PageDependencies.Add(FixupPageIndex);
            }
            P age S streaming S tate . D ependencies N um PageStreamingState.DependenciesStart;                    =     R   es

      }

      //deal withpage.
      structFPageResult
      {
            TArray<uint8> Data;
            uint32 UncompressedSize;
      };
      TArray< FPageResult > PageResults;
      PageResults.SetNum(NumPages);

      //Parallel processing
      ParallelFor(NumPages, [&Resources, &Pages, &Groups, &Parts, &Clusters, &EncodingInfos, &FixupChunks,
&PageResults, NumTexCoords, bLZCompress](int32 PageIndex)
      {
            constFPage& Page = Pages[PageIndex]; FFixupChunk&
            FixupChunk = FixupChunks[PageIndex];

            //IncreasehierarchyCorrection.
            {
                  // Parts include the hierarchy fixups for all the other parts of the same
group.
                  uint32 NumHierarchyFixups =0;
                  for(uint32 i =0; i < Page.PartsNum; i++) {

                        constFClusterGroupPart& Part = Parts[Page.PartsStartIndex + i]; constFClusterGroup&
                        Group = Groups[Part.GroupIndex];
```

```
                const uint32 HierarchyRootOffset =
Resources.HierarchyRootOffsets[Group.MeshIndex];

                uint32 PageDependencyStart = Group.PageIndexStart; uint32 PageDependencyNum =
                Group.PageIndexNum; RemoveRootPagesFromRange(PageDependencyStart,
                PageDependencyNum);

                // Add fixups to all parts of the group for(uint32 j =0; j <
                Group.PageIndexNum; j++) {

                    const FPage& Page2 = Pages[Group.PageIndexStart + j]; for(uint32 k =0;
                    k < Page2.PartsNum; k++) {

                        const FClusterGroupPart& Part2 = Parts[Page2.PartsStartIndex + k]; if(Part2.GroupIndex
                        == Part.GroupIndex) {

                            const uint32 GlobalHierarchyNodeIndex = HierarchyRootOffset +
Part2.HierarchyNodeIndex;
                            FixupChunk.GetHierarchyFixup(NumHierarchyFixups++) =
FHierarchyFixup(Part2.PageIndex, GlobalHierarchyNodeIndex, Part2.HierarchyChildIndex, Part2.PageClusterOffset,
PageDependencyStart, PageDependencyNum);
                            break;
                        }
                    }
                }
            }
            check ( N um H ierarchy F ixups                                    = =      F ixup C hu                n
        }

        // Pack clusters and generate material range data TArray<uint32>
                                                CombinedStripBitmaskData;
        TArray<uint32>                          CombinedVertexRefBitmaskData;
        TArray<uint32>                          CombinedVertexRefData;
        TArray<uint8>                          CombinedIndexData;
        TArray<uint8>                          CombinedPositionData;
        TArray<uint8>                          CombinedAttributeData;
        TArray<uint32>                          MaterialRangeData;
        TArray<uint16>                          CodedVerticesPerCluster;
        TArray<uint32>                          NumVertexBytesPerCluster;
        TArray<FPackedCluster>                  PackedClusters;

        PackedClusters.SetNumUninitialized(Page.NumClusters);
        CodedVerticesPerCluster.SetNumUninitialized(Page.NumClusters);
        NumVertexBytesPerCluster.SetNumUninitialized(Page.NumClusters);

        const uint32 NumPackedClusterDwords = Page.NumClusters *sizeof(FPackedCluster) / sizeof(uint32);


        FPageSections GpuSectionOffsets = Page.GpuSizes.GetOffsets();
        TMap<FVariableVertex, uint32> UniqueVertices;

        for(uint32 i =0; i < Page.PartsNum; i++) {

            const FClusterGroupPart& Part = Parts[Page.PartsStartIndex + i]; for(uint32 j =0; j <
            (uint32)Part.Clusters.Num(); j++) {

                const uint32 ClusterIndex = Part.Clusters[j]; const FCluster& Cluster
                = Clusters[ClusterIndex];
```

```
                    constFEncodingInfo& EncodingInfo = EncodingInfos[ClusterIndex];

                    constuint32 LocalClusterIndex = Part.PageClusterOffset + j; FPackedCluster&
                    PackedCluster = PackedClusters[LocalClusterIndex]; PackCluster(PackedCluster, Cluster,
                    EncodingInfos[ClusterIndex],
NumTexCoords);

                    PackedCluster.PackedMaterialInfo = PackMaterialInfo(Cluster,
MaterialRangeData, NumPackedClusterDwords);
                    check((GpuSectionOffsets.Index &3) ==0); check((GpuSectionOffsets.Position &3) ==0);
                    check((GpuSectionOffsets.Attribute &3) ==0;









                    GpuSectionOffsets += EncodingInfo.GpuSizes;

                    constuint32 PrevVertexBytes = CombinedPositionData.Num(); uint32
                    NumCodedVertices =0;
                    EncodeGeometryData( LocalClusterIndex, Cluster, EncodingInfo,
NumTexCoords,
                                        CombinedStripBitmaskData, CombinedIndexData,
                                        CombinedVertexRefBitmaskData, CombinedVertexRefData,
CombinedPositionData, CombinedAttributeData,
                                        UniqueVertices, NumCodedVertices);

                    NumVertexBytesPerCluster[LocalClusterIndex] = CombinedPositionData.Num() -
PrevVertexBytes;
                    CodedVerticesPerCluster[LocalClusterIndex]          = NumCodedVertices;
                }
}
check ( G pu Section O ffsets . C luster                                        ==

P ag ech.ecGk(p     AliugnS(GpiuSzecetiosnO.ffGseets.tMaterialTable,16)          ==
PMagae.tGpeuSirzeis.aGetlDeTcoadeIbnfloOeffsOetf()f) ; s
et ( )ch)ec;k(GpuSectionOffsets.DecodeInfo                                      ==
Page.GpuSizes.GetIndexOffset());
            check(GpuSectionOffsets.Index                                       ==
            Page.GpuSizes.GetPositionOffset());
            check(GpuSectionOffsets.Position                                    ==
            Page.GpuSizes.GetAttributeOffset());
            check(GpuSectionOffsets.Attribute                                   == Page.GpuSizes.GetTotal());
            // DwordAlign index data.
            CombinedIndexData.SetNumZeroed((CombinedIndexData.Num() +3) &-4);


            //Directly inPackkedclustersPerform internal repairs on the page.
            for(uint32 LocalPartIndex =0; LocalPartIndex < Page.PartsNum; LocalPartIndex++) {

                constFClusterGroupPart& Part = Parts[Page.PartsStartIndex + LocalPartIndex]; constFClusterGroup&
                Group = Groups[Part.GroupIndex]; uint32 GeneratingGroupIndex = MAX_uint32;

                for(uint32 ClusterPositionInPart =0; ClusterPositionInPart <
(uint32)Part.Clusters.Num(); ClusterPositionInPart++)
                {
                    constFCluster& Cluster = Clusters[Part.Clusters[ClusterPositionInPart]];
```

```
                    if(Cluster.GeneratingGroupIndex != INVALID_GROUP_INDEX) {

                        constFClusterGroup& GeneratingGroup =
Groups[Cluster.GeneratingGroupIndex];
                        uint32 PageDependencyStart = Group.PageIndexStart; uint32 PageDependencyNum =
                        Group.PageIndexNum; RemoveRootPagesFromRange(PageDependencyStart,
                        PageDependencyNum);

                        if(GeneratingGroup.PageIndexStart == PageIndex &&
GeneratingGroup.PageIndexNum ==1)
                        {
                            //currentPageDependencies that have already been satisfied, just fix
                            them. PackedClusters[Part.PageClusterOffset +
ClusterPositionInPart].Flags &= ~NANITE_CLUSTER_FLAG_LEAF;// Mark parent as no longer leaf

                        }
                    }
                }
            }

            //startpage
            FPageResult& PageResult = PageResults[PageIndex];
            PageResult.Data.SetNum(CLUSTER_PAGE_DISK_SIZE); FBlockPointer
            PagePointer(PageResult.Data.GetData(),                    PageResult.Data.Num());

            //Disk header information.
            FPageDiskHeader* PageDiskHeader = PagePointer.Advance<FPageDiskHeader>(1);

            // 16Byte-aligned material range data, making it easy toGPUCopied during transcoding.
            MaterialRangeData.SetNum(Align(MaterialRangeData.Num(),4));

            static_assert(sizeof(FUVRange) %16==0,"sizeof(FUVRange) must be a multiple of
16");
            static_assert(sizeof(FPackedCluster) %16==0,"sizeof(FPackedCluster) must be a multiple of 16");

            PageDiskHeader->NumClusters = Page.NumClusters; PageDiskHeader->GpuSize =
            Page.GpuSizes.GetTotal(); PageDiskHeader->NumRawFloat4s = Page.NumClusters * (sizeof
            (FPackedCluster) + NumTexCoords *sizeof(FUVRange)) /16+ MaterialRangeData.Num() /4;

            PageDiskHeader->NumTexCoords = NumTexCoords;

            // ClusterHeader information.
            FClusterDiskHeader* ClusterDiskHeaders = PagePointer.Advance<FClusterDiskHeader> (Page.NumClusters);


            //UseSOA (Structure-of-Arrays)Memory layout writingcluster. {

                constuint32 NumClusterFloat4Propeties =sizeof(FPackedCluster) /16; for(uint32 float4Index =
                0; float4Index < NumClusterFloat4Propeties;
float4Index++)
                {
                    for(constFPackedCluster& PackedCluster : PackedClusters) {

                        uint8* Dst = PagePointer.Advance<uint8>(16); FMemory::Memcpy(Dst,
                        (uint8*)&PackedCluster + float4Index *16,16);
                    }
                }
            }
```

```
//Material table.
uint32 MaterialTableSize = MaterialRangeData.Num() *
MaterialRangeData.GetTypeSize();
uint8* MaterialTable = PagePointer.Advance<uint8>(MaterialTableSize); FMemory::Memcpy(MaterialTable,
MaterialRangeData.GetData(), MaterialTableSize); check(MaterialTableSize ==
Page.GpuSizes.GetMaterialTableSize());

//Decode the information.
PageDiskHeader->DecodeInfoOffset = PagePointer.Offset(); for(uint32 i =0; i
< Page.PartsNum; i++) {

    constFClusterGroupPart& Part = Parts[Page.PartsStartIndex + i]; for(uint32 j =0; j <
    (uint32)Part.Clusters.Num(); j++) {

        constuint32 ClusterIndex = Part.Clusters[j];
        FUVRange* DecodeInfo = PagePointer.Advance<FUVRange>(NumTexCoords); for(uint32 k
        =0; k < NumTexCoords; k++) {

            DecodeInfo[k] = EncodingInfos[ClusterIndex].UVInfos[k].UVRange;
        }
    }
}
//Index data.
{

    uint8* IndexData = PagePointer.GetPtr<uint8>();
#if  USE_STRIP_INDICES
    for(uint32 i =0; i < Page.PartsNum; i++) {

        constFClusterGroupPart& Part = Parts[Page.PartsStartIndex + i]; for(uint32 j =0; j <
        (uint32)Part.Clusters.Num(); j++) {

            constuint32 LocalClusterIndex = Part.PageClusterOffset + j; constuint32
            ClusterIndex = Part.Clusters[j]; constFCluster& Cluster = Clusters[ClusterIndex];


            ClusterDiskHeaders[LocalClusterIndex].IndexDataOffset =
PagePointer.Offset();
            ClusterDiskHeaders[LocalClusterIndex].NumPrevNewVerticesBeforeDwords                =
Cluster.StripDesc.NumPrevNewVerticesBeforeDwords;
            ClusterDiskHeaders[LocalClusterIndex].NumPrevRefVerticesBeforeDwords                =
Cluster.StripDesc.NumPrevRefVerticesBeforeDwords;

            PagePointer.Advance<uint8>(Cluster.StripIndexData.Num());
        }
    }

    uint32 IndexDataSize = CombinedIndexData.Num() *
CombinedIndexData.GetTypeSize();
    FMemory::Memcpy(IndexData, CombinedIndexData.GetData(),          IndexDataSize);
    PagePointer.Align(sizeof(uint32));

    PageDiskHeader->StripBitmaskOffset = PagePointer.Offset(); uint32
    StripBitmaskDataSize = CombinedStripBitmaskData.Num() *
CombinedStripBitmaskData.GetTypeSize();
    uint8* StripBitmaskData = PagePointer.Advance<uint8>(StripBitmaskDataSize);
    FMemory::Memcpy(StripBitmaskData, CombinedStripBitmaskData.GetData(),
```

```
StripBitmaskDataSize);

#else
                for(uint32 i =0; i < Page.NumClusters; i++) {

                        ClusterDiskHeaders[i].IndexDataOffset = PagePointer.Offset();
                        PagePointer.Advance<uint8>(PackedClusters[i].GetNumTris() *3);
                }
                Page Pointer . Align (sizeof( uint 3 2 ) ) ;

                uint32 IndexDataSize = CombinedIndexData.Num() *
CombinedIndexData.GetTypeSize();
                FMemory::Memcpy(IndexData, CombinedIndexData.GetData(),                 IndexDataSize);
#endif
        }

        //Bitmask to write vertex references to.

        {
                PageDiskHeader->VertexRefBitmaskOffset = PagePointer.Offset();
                constuint32 VertexRefBitmaskSize = Page.NumClusters * (MAX_CLUSTER_VERTICES /
8);
                uint8* VertexRefBitmask = PagePointer.Advance<uint8>(VertexRefBitmaskSize);
                FMemory::Memcpy(VertexRefBitmask, CombinedVertexRefBitmaskData.GetData(),
VertexRefBitmaskSize);
                check(CombinedVertexRefBitmaskData.Num() *
CombinedVertexRefBitmaskData.GetTypeSize() == VertexRefBitmaskSize);
        }

        //Write vertex references.

        {
                uint8* VertexRefs = PagePointer.GetPtr<uint8>(); for(uint32 i =0; i
                < Page.NumClusters; i++) {

                        ClusterDiskHeaders[i].VertexRefDataOffset = PagePointer.Offset(); uint32
                        NumVertexRefs = PackedClusters[i].GetNumVerts() -
CodedVerticesPerCluster[i];
                        PagePointer.Advance<uint32>(NumVertexRefs);
                }
                FM emory : : M emcpy ( V ertex R efs ,                                                  Compined                V   e
CombinedVertexRefData.Num() * CombinedVertexRefData.GetTypeSize());
        }
        //Write location.
        {

                uint8* PositionData = PagePointer.GetPtr<uint8>(); for(uint32 i =0; i
                < Page.NumClusters; i++) {

                        ClusterDiskHeaders[i].PositionDataOffset = PagePointer.Offset();
                        PagePointer.Advance<uint8>(NumVertexBytesPerCluster[i]);
                }
                check (                 ( P age P ointer . G et P tr < uint 8 > ( ) CombinedPositionData.GetTypeSize());                -
CombinedPositionData.Num()        *

                FMemory::Memcpy(PositionData, CombinedPositionData.GetData(),
CombinedPositionData.Num() * CombinedPositionData.GetTypeSize());
        }

        //Write attributes.
```

```
        {
            uint8* AttribData = PagePointer.GetPtr<uint8>(); for(uint32 i =0; i
            < Page.NumClusters; i++) {

                constuint32 BytesPerAttribute = (PackedClusters[i].GetBitsPerAttribute()
+  7) /8;
                ClusterDiskHeaders[i].AttributeDataOffset = PagePointer.Offset();
                PagePointer.Advance<uint8>(Align(CodedVerticesPerCluster[i] *
BytesPerAttribute,4));
            }
            check ( ( uint 3 2 ) ( P age Pointer . G et P tr < ui                                                    nt
CombinedAttributeData.Num() * CombinedAttributeData.GetTypeSize());
            FMemory::Memcpy(AttribData, CombinedAttributeData.GetData(),
CombinedAttributeData.Num()* CombinedAttributeData.GetTypeSize());
        }

        //   UsageLempel-Ziv (LZ)Lossless compressed memory,LZA variation ofLempel-Ziv-Welch
        //   (LZW). More details:http://athena.ecs.csus.edu/~wang/DLZW.pdf. (bLZCompress)
        if
        {
            TArray<uint8> DataCopy(PageResult.Data.GetData(), PagePointer.Offset());
            PageResult.UncompressedSize = DataCopy.Num();

            int32 CompressedSize = PageResult.Data.Num();
            verify(FCompression::CompressMemory(NAME_LZ4,          PageResult.Data.GetData(),
CompressedSize, DataCopy.GetData(), DataCopy.Num()));

            PageResult.Data.SetNum(CompressedSize,false);
        }
        else            //          Do not use compression.
        {
            PageResult.Data.SetNum(PagePointer.Offset(),false);
            PageResult.UncompressedSize = PageResult.Data.Num();
        }
    });

    //WritePage.
    uint32 TotalUncompressedSize =0; uint32
    TotalCompressedSize =0; uint32
    TotalFixupSize =0;
    for(uint32 PageIndex =0; PageIndex < NumPages; PageIndex++) {

        constFPage& Page = Pages[PageIndex];

        FFixupChunk& FixupChunk = FixupChunks[PageIndex];
        TArray<uint8>& BulkData = IsRootPage(PageIndex) ? Resources.RootClusterPage : StreamableBulkData;


        FPageStreamingState& PageStreamingState =
        Resources.PageStreamingStates[PageIndex];
        PageStreamingState.BulkOffset = BulkData.Num();

        //Write the correction block.

        uint32  FixupChunkSize  =  FixupChunk.GetSize();
        check(FixupChunk.Header.NumHierachyFixups          <MAX_CLUSTERS_PER_PAGE);
        check(FixupChunk.Header.NumClusterFixups          <    MAX_CLUSTERS_PER_PAGE);
        BulkData.Append((uint8*)&FixupChunk, FixupChunkSize); TotalFixupSize
        += FixupChunkSize;
```

```
            //Copy page toBulkData.
            TArray<uint8>& PageData = PageResults[PageIndex].Data;
            BulkData.Append(PageData.GetData(),                PageData.Num());
            TotalUncompressedSize         + = PageResults[PageIndex].UncompressedSize;
            TotalCompressedSize +=        PageData.Num();

            PageStreamingState.BulkSize = BulkData.Num() - PageStreamingState.BulkOffset;
            PageStreamingState.PageUncompressedSize = PageResults[PageIndex].UncompressedSize;
        }

        uint32 TotalDiskSize = Resources.RootClusterPage.Num() + StreamableBulkData.Num(); UE_LOG(LogStaticMesh,
        Log, TEXT("WritePages:"), NumPages); UE_LOG(LogStaticMesh, Log, TEXT("%d pages written."), NumPages);
        UE_LOG(LogStaticMesh, Log, TEXT("GPU size: %d bytes. %.3f bytes per page. %.3f%% utilization."),
        TotalGPUSize, TotalGPUSize /float(NumPages), TotalGPUSize / (float(NumPages) * CLUSTER_PAGE_GPU_SIZE) *
100.0f);

        UE_LOG(LogStaticMesh, Log, TEXT("Uncompressed page data: %d bytes. Compressed page data: %d bytes. Fixup
data: %d bytes."), TotalUncompressedSize, TotalCompressedSize, TotalFixupSize);

        UE_LOG(LogStaticMesh, Log, TEXT("Total disk size: %d bytes. %.3f bytes per page."), TotalDiskSize, TotalDiskSize/
float(NumPages));

        //storagePageData.
        Resources.StreamableClusterPages.Lock(LOCK_READ_WRITE); uint8* Ptr =

(uint8*)Resources.StreamableClusterPages.Realloc(StreamableBulkData.Num());
        FMemory::Memcpy(Ptr, StreamableBulkData.GetData(), StreamableBulkData.Num());
        Resources.StreamableClusterPages.Unlock();
        Resources.StreamableClusterPages.SetBulkDataFlags(BULKDATA_Force_NOT_InlinePayload);
        Resources.bLZCompressed = bLZCompress;
}
```

6.4.2.8 FImposterAtlas::Rasterize

```
// Engine\Source\Developer\NaniteBuilder\Private\ImposterAtlas.cpp

//Will specifyClusterRasterization toImposter.
void FImposterAtlas::Rasterize(constFIntPoint& TilePos,constFCluster& Cluster, uint32 ClusterIndex )

{
        constexpr uint32 ViewSize = TileSize;// * SuperSample;

        FIntRectScissor(0,0, ViewSize, ViewSize );

        //Get local toImposterThe transformation matrix of .
        FMatrix LocalToImposter = GetLocalToImposter( TilePos );

        TArray< FVector, TInlineAllocator<128> > Positions;
        Positions.SetNum( Cluster.NumVerts,false);

        //extractClusterVertex position, and transform toImposterspace.
        for( uint32 VertIndex =0; VertIndex < Cluster.NumVerts; VertIndex++ ) {

            FVector Position = Cluster.GetPosition( VertIndex ); Position =
            LocalToImposter.TransformPosition( Position );
```

```
        Positions[VertIndex].X = (Position.X *0.5f+0.5f) * ViewSize; Positions[ VertIndex ].Y = ( Position.Y *0.5f+
        0.5f) * ViewSize; Positions[ VertIndex ].Z = ( Position.Z *0.5f+0.5f) *254.0f+1.0f; zero is reserved as
        masked                                                                                                  //

    }

    //Iterate over all triangles and rasterize them toImposter.
    for( uint32 TriIndex =0; TriIndex < Cluster.NumTris; TriIndex++ ) {

        FVectorVerts[3]; Verts[0] = Positions[ Cluster.Indexes[ TriIndex *3+0] ]; Verts[1] =
        Positions[ Cluster.Indexes[ TriIndex *3+1] ]; Verts[2] =
        Positions[ Cluster.Indexes[ TriIndex *3+2] ];


        //Rasterize triangles.
        RasterizeTri(Verts, Scissor,0,
            //Save the rasterized result.
            [&]( int32 x, int32 y,floatz ) {

                uint32 Depth = FMath::RoundToInt( FMath::Clamp( z,1.0f,255.0f) ); uint16 PixelValue = ( Depth
                <<8) | ( ClusterIndex <<7) | TriIndex; //uint32 PixelIndex = x + y * ViewSize;

                uint32 PixelIndex = x + ( y + ( TilePos.X + TilePos.Y * AtlasSize ) *
TileSize ) * TileSize;
                Pixels[ PixelIndex ] = FMath::Max( Pixels[ PixelIndex ], PixelValue );
            } );
    }
}

// Engine\Source\Developer\NaniteBuilder\Private\Rasterizer.h

//The triangle specified by the soft raster is called when writing data.FWritePixelCallback function.

template<typename FWritePixel>
void RasterizeTri(constFVectorVerts[3],constFIntRect& ScissorRect, uint32 SubpixelDilate, FWritePixel
WritePixel )
{
    constexpr uint32 SubpixelBits constexpr          =8;
    uint32 SubpixelSamples                           =1<< SubpixelBits;

    FVector v01 = Verts[1] - Verts[0]; FVector v02 =
    Verts[2] - Verts[0];

    floatDetXY = v01.X * v02.Y - v01.Y * v02.X; if( DetXY >=0.0f) {


        //Backface culling.
        //If not culled, need to swap vertices, correct for the rest of the code
        winding. return;

    }

    FVector2D GradZ; GradZ.X = ( v01.Z * v02.Y - v01.Y * v02.Z ) / DetXY; ;



    // 24.8 fixed pointFIntPoint Vert0 = ToIntPoint( Verts[0] * SubpixelSamples );
    FIntPoint Vert1 = ToIntPoint( Verts[1] * SubpixelSamples ); FIntPoint Vert2 =
    ToIntPoint( Verts[2] * SubpixelSamples );
```

```
/M oment FIshapeBoundingBox RectSubpixel(Vert0, Vert0);
                RectSubpixel.Include(       Vert1               ) ;
RectSubpixel.Include(                       Vert2               ) ;
RectSubpixel.InflateRect(               SubpixelDilate        ) ;


//Round to the nearest pixel.
FIntRect RectPixel = ( ( RectSubpixel + (SubpixelSamples /2) -1) ) / SubpixelSamples;


//Crop to viewport.
RectPixel.Clip( ScissorRect );

//If no pixel is covered, crop it.
if( RectPixel.IsEmpty() )
        return;

// 12.8 fixed pointFIntPoint Edge01 = Vert0 -
Vert1; FIntPoint Edge12 = Vert1 - Vert2;
FIntPoint Edge20 = Vert2 - Vert0;


//Adjust with half-pixel offsetMinPixel.
// 12.8 fixed point
//Maximum triangle size =2047x2047Pixel.
constFIntPoint BaseSubpixel = RectPixel.Min * SubpixelSamples + (SubpixelSamples /
2);
Vert0    - =  BaseSubpixel;
Vert1    - =  BaseSubpixel;
Vert2    - =  BaseSubpixel;

autoEdgeC = [=](constFIntPoint& Edge,constFIntPoint& Vert ) {

        int64 ex = Edge.X; int64
        ey = Edge.Y; int64 vx =
        Vert.X; int64 vy = Vert.Y;


        // Half-edge constants
        // 24.16 fixed point int64 C = ey * vx -
        ex * vy;

        //Corrected Fill Convention (fill
        convention) // Top left rule for CCW
        C -= (Edge.Y <0|| (Edge.Y ==0&& Edge.X >0) ) ?0:1;

        //Expand the side.
        C += ( FMath::Abs( Edge.X ) + FMath::Abs( Edge.Y ) ) * SubpixelDilate;

        //      Pixel increment step.
        //      The low order bits are always the same and therefore do not matter when testing for signs.
        // 24.8 fixed point
        returnint32( C >> SubpixelBits );
};

int32 C0 = EdgeC( Edge01, Vert0 ); int32 C1 =
EdgeC( Edge12, Vert1 );
```

```
    int32 C2 = EdgeC( Edge20, Vert2 );
    floatZ0 = Verts[0].Z - ( GradZ.X * Vert0.X + GradZ.Y * Vert0.Y ) / SubpixelSamples;

    int32 CY0 = C0;
    int32 CY1 = C1;
    int32 CY2 = C2;
    floatZY = Z0;

    //Traverse all pixels in the rectangle and fill the pixels in the triangle.
    for( int32 y = RectPixel.Min.Y; y < RectPixel.Max.Y; y++ ) {

        int32 CX0 = CY0; int32
        CX1 = CY1; int32 CX2 =
        CY2; floatZX = ZY;


        for( int32 x = RectPixel.Min.X; x < RectPixel.Max.X; x++ ) {

            //If the current3SideXThe components are all positive numbers, indicating that within the
            triangle, the callWritePixel if( ( CX0 | CX1 | CX2 ) >=0) {

                WritePixel( x, y, ZX );
            }

            CX0  - =  Edge01.Y;
            CX1  - =  Edge12.Y;
            CX2  - =  Edge20.Y;
            ZX  += GradZ.X;
        }

        CY0  + =  Edge01.X;
        CY1  + =  Edge12.X;
        CY2  + =  Edge20.X;
        ZY  += GradZ.Y;
    }
}
```

Write data.

6.4.2.9 Summary of Nanite Data Construction

This section summarizes the process of building Nanite data. The initial entry is **BuildNaniteFromHiResSourceModel**:

- Get Nanite's high-precision model from UStaticMesh's HiResSourceModel.
- Calculate tangents, light map UVs, etc.
- Build temporary RenderData data to pass to subsequent Nanite construction stages.
- Build per-section index, vertex, and index buffers.
- BuildCombinedSectionIndices: Combine section-by-section index buffers.
- ComputeBoundsFromVertexList: Computes bounding boxes from a high-resolution mesh before Nanite build.
- Parse the Section material index from SectionInfoMap.
- **NaniteBuilderModule.Build**:Executes the Nanite build module.

The following is an overview of the main process of**NaniteBuilderModule.Build**:

- Build an associative array of triangle indices and material indices.
- **BuildNaniteData**:Build Nanite data.
    - Processing vertex colors.
    - Traverse all Sections and build a Cluster for each Section.
        - **ClusterTriangles**:Split a Section into one or more Clusters.
            - Initialize shared edges, boundary edges, edge hashes and other data.
            - Process edge hashing in parallel.
            - Find shared edges and border edges in parallel.
            - Handling of disjoint triangle sets.
            - Use FGraphPartitioner to partition the grid.
                - FGraphPartitioner uses the METIS third-party open source library. METIS can efficiently provide high-quality mesh partitioning, and has the characteristics of low filling rate, which can ensure the effect and efficiency of mesh partitioning. The partitioning algorithm of METIS has three stages: coarsening, partitioning, and uncoarsening.
            - Build Clusters in parallel.
    - Checks whether the original static mesh data needs to be replaced with a coarse representation.
        - The use of rough representation requires that the PercentTriangles set by Nanite is less than 1 and the number of triangles in the original mesh is greater than 2000.
    - **Call BuildDAG**for all Sections to build a directed acyclic graph to accelerate surface and module reduction.
    - If a coarse representation is used,**BuildCoarseRepresentation**is called to build the coarse representation data, and then the coarse mesh range is used to correct the mesh section information while respecting the original sequence number and preserving the material.
    - **Encode**:Encodes a Nanite grid.
        - RemoveDegenerateTriangles: Delete all degenerate triangles of Cluster.
        - BuildMaterialRanges: Build material ranges for all Clusters.
        - ConstrainClusters: Constrain Cluster to ClusterGroup.
        - CalculateQuantizedPositionsUniformGrid: Calculates quantized positions.
        - CalculateEncodingInfos: Calculate encoding information.
        - AssignClustersToPages: Assign Cluster to Page. BuildHierarchies:
        - Build the hierarchical nodes of ClusterGroup. WritePages: Writes
        - Cluster and ClusterGroup information to Page.
    - If necessary (when there is only one Section), generate FImposterAtlas.
        - Generate Imposters for all Clusters in parallel and rasterize all triangles of each Cluster into FImposterAtlas.

A lot of optimization techniques are used in the construction process of Nanite, mainly including but not limited to:

- ParallelFor is used extensively to process logic in parallel using multiple threads and reduce build time.
- Excellent meshing can be obtained using MITES.
- Use concepts of different levels such as Cluster, ClusterGroup, and ClusterGroupPart to organically combine grid-related data.
- Use DAG, Hierarchy, Mip Level, Coarse Representation, etc. to speed up and optimize mesh construction and division.
- Generate the Page, ImposterAtlas and other information required for GPU rendering in advance.
- A lot of high-level data compression is used, such as LZ Compression, fixed point, bit operations, etc.

In addition, Nanite does not use the previously rumored Geometry Image technology, but the core idea or technology is still similar.

> It is said thatamong the UE5 coders there is an author who has published papers with Professor Gu Xianfeng, a disciple of international mathematics master Qiu Chengtong and a pioneer of geometric graphics.
> Regarding Geometry Image technology, you can refer to Professor Gu's paperGeometry imagesand his public account: Lao Gu Talks Geometry.

6.4.3 Nanite Rendering

This section will explain the code and logic of the Nanite rendering stage.

6.4.3.1 Nanite Rendering Overview

UE5 has made major changes to the rendering module to support the rendering of Nanite features, which can be summarized as follows:

- **Engine Modules:**
  - Added FMeshNaniteSettings type, which has been analyzed in the previous section.
  - FStaticMeshRenderData adds Nanite::FResources data and related processing logic.
  - UStaticMeshComponent added bDisplayNaniteProxyMesh and added FMeshNaniteSettings data.
  - Components such as UHierarchicalInstancedStaticMeshComponent and UInstancedStaticMeshComponent add support for Nanite data and SceneProxy.
  - Added InstanceUniformShaderParameters module.
- **Rendering module:**

- Added NaniteRender module, including FNaniteCommandInfo, ENaniteMeshPass, FNaniteDrawListContext, FCullingContext, FRasterContext, FRasterResults, FNaniteShader, FNaniteMaterialVS, FNaniteMeshProcessor, FNaniteMaterialTables, ERasterTechnique, ERasterScheduling, EOutputBufferMode, FPackedView and other types and processing interfaces.
- FPrimitiveSceneInfo adds NaniteCommandInfos, NaniteMaterialIds, LumenPrimitiveIndex, and CachedRayTracingMeshCommandsHashPerLOD, bRegisteredWithVelocityData, InstanceDataOffset, NumInstanceDataEntries instantiation and ray tracing related data and processing interfaces.
- Added NaniteResources module, including Nanite::FSceneProxy, Nanite::FResources, Nanite::FVertexFactory and other types.
- Added NaniteStreamingManager module, including Nanite::FPageKey, Nanite::FGPUStreamingRequest, Nanite::FStreamingRequest, Nanite::FStreamingPageInfo, Nanite::FRootPageInfo, Nanite::FPendingPage, Nanite::FAsyncState, Nanite::FStreamingManager, Nanite::, Nanite:: and other types. FPrimitiveSceneProxyzIncreaseSupportsNaniteRendering,IsNaniteMesh, bSuppor tsMeshCardRepresentation,IsAlwaysVisible,GetPrimitiveInstances, RayTracingGroupIdwait.
- SceneInterface and SceneManagement add types such as FInstanceCullingManagerResources, and use FGPUScenePrimitiveCollector instead of FPrimitiveUniformShaderParameters.
- SceneViewIncreaseFViewShaderParameters,PrecomputedIndirectLightingColorScale, GlobalDistanceField,VirtualTexture,PhysicsField,Lumen,Instance,PagewaitshaderBinding.
- Added bIsNaniteMesh flag to FPrimitiveFlagsCompact and other types.

- **Shader module:**
  - AddedClusterCulling,Culling,HZBCull,InstanceCulling,MaterialCulling,Shadow, GBuffer, Imposter,DataDecode,DataPacked,Rasterizer,WritePixelwait.

6.4.3.2 Nanite rendering basics

This section analyzes the main concepts, types, and interfaces involved in Nanite rendering.

- **InstanceUniformShaderParameters**

```
// Engine\Source\Runtime\Engine\Public\InstanceUniformShaderParameters.h

#define	INSTANCE_SCENE_DATA_FLAG_CAST_SHADOWS		0x1
#define	INSTANCE_SCENE_DATA_FLAG_DETERMINANT_SIGN	0x2
#define	INSTANCE_SCENE_DATA_FLAG_HAS_IMPOSTER		0x4
```

```
// NaniteInstantiation information.
classFNaniteInfo
{
public:

    uint32 RuntimeResourceID;//The resource identifier at runtime.
    uint32 HierarchyOffset_AndHasImposter;//Hierarchy offset and whether there isImposterof joint data.

    FNaniteInfo()                                   :
    RuntimeResourceID(0xFFFFFFFFu)                  ,
    HierarchyOffset_AndHasImposter(0xFFFFFFFFu) }   {



    FNaniteInfo(uint32 InRuntimeResourceID, int32 InHierarchyOffset,boolbHasImposter) :
    RuntimeResourceID(InRuntimeResourceID)                                                ,
    HierarchyOffset_AndHasImposter((InHierarchyOffset <<1) | (bHasImposter ?1u:0u)) { }



} ;

// NanitePrimitive instantiation information.

structFPrimitiveInstance {

    FMatrix    InstanceToLocal;
    FMatrix      FMatrix PrevInstanceToLocal;
    FVector4   LocalToWorld;
    FBoxSphereBounds PrevLocalToWorld;
    FBoxSphereBounds NonUniformScale;
                InvNonUniformScaleAndDeterminantSign;
                        RenderBounds;
                        LocalBounds;
    FVector4 LightMapAndShadowMapUVBias; uint32
    PrimitiveId;//GraphicsID. FNaniteInfo NaniteInfo;//
    Naniteinformation. uint32
                LastUpdateSceneFrameNumber;
    float       PerInstanceRandom;
    uint32      Flags;
};

(.....)

// FInstanceUniformShaderParametersStatement. Need and shaderofFInstanceSceneDataStrict match.
BEGIN_GLOBAL_SHADER_PARAMETER_STRUCT(FInstanceUniformShaderParameters,ENGINE_API)
    SHADER_PARAMETER(FMatrix,    LocalToWorld)
    SHADER_PARAMETER(FVector4,   PrevLocalToWorld)
    SHADER_PARAMETER(FVector4,     NonUniformScale)
    SHADER_PARAMETER(FVector,      InvNonUniformScaleAndDeterminantSign)
    SHADER_PARAMETER(uint32,       LocalBoundsCenter)
    SHADER_PARAMETER(FVector,      PrimitiveId)
    SHADER_PARAMETER(uint32,     LocalBoundsExtent)
    SHADER_PARAMETER(uint32,       LastUpdateSceneFrameNumber)
    SHADER_PARAMETER)              ,  NaniteRuntimeResourceID)
    SHADER_PARAMETER(float,        NaniteHierarchyOffset)
    SHADER_PARAMETER(uint32,       PerInstanceRandom
    SHADER_PARAMETER(FVector4,     Flags

    END_GLOBAL_SHADER_PARAMETER_LSiTgRhUtMCTa(p)AndShadowMapUVBias)
```

```
//Instantiate scene coloring data.
struct FInstanceSceneShaderData
{
    // Need andSceneData.ushofGetInstanceData()Matches.
    enum{InstanceDataStrideInFloat4s =10};

    FVector4 Data[InstanceDataStrideInFloat4s];

    (.....)
};
```

- **NaniteStreamingManager**

```
// Engine\Source\Runtime\Engine\Public\Rendering\NaniteStreamingManager.h

namespace Nanite
{

//Page Key Value
struct FPageKey
{
    // Runtime ResourcesID.
    uint32 RuntimeResourceID;
    //Page index.
    uint32 PageIndex;
};

//Key-Value Hashing
FORCEINLINE uint32GetTypeHash(constFPageKey& Key ) {

    returnKey.RuntimeResourceID *0xFC6014F9u + Key.PageIndex *0x58399E77u;
}

//Key value comparison.
FORCEINLINEbooloperator==(constFPageKey& A,constFPageKey& B ) {

    returnA.RuntimeResourceID == B.RuntimeResourceID && A.PageIndex == B.PageIndex;
}
FORCEINLINEbooloperator!=(constFPageKey& A,constFPageKey& B) {

    return!(A == B);
}

//Deduplicationdeduplication 【Before】 data information.
structFGPUStreamingRequest {

    uint32          RuntimeResourceID;
    uint32          PageIndex_NumPages;
    uint32          Priority;
};

//Deduplicationdeduplication 【After】 data information.
structFStreamingRequest {

    FPageKey    Key;
    uint32          Priority;
};
```

```
//Streaming page information.
structFStreamingPageInfo {

    FStreamingPageInfo*     Next;
    FStreamingPageInfo*     Prev;

    FPageKey        RegisteredKey;
    FPageKey        ResidentKey;

    uint32              GPUPageIndex;
    uint32              LatestUpdateIndex;
    uint32              RefCount;
};

//Root page information.
structFRootPageInfo
{
    uint32      RuntimeResourceID;
    uint32      NumClusters;
};

//Suspend page.
structFPendingPage
{
#if!WITH_EDITOR
    uint8*                      MemoryPtr;
    FIoRequest                  Request;
    IAsyncReadFileHandle*       AsyncHandle;
    IAsyncReadRequest*          AsyncRequest;
#endif

    uint32                      GPUPageIndex;
    FPageKey                    InstallKey;
#if !UE_BUILD_SHIPPING
    uint32                      BytesLeftToStream;
#endif
};

//Asynchronous information.
structFAsyncState
{
    FRHIGPUBufferReadback*      LatestReadbackBuffer        = nullptr;
    const uint32*   uint32      LatestReadbackBufferPtr     = nullptr;
    bool bool                   NumReadyPages               = 0;
                                bUpdateActive               = false;
                                bBuffersTransitionedToWrite = false;
};

// NaniteStream Manager.
classFStreamingManager: public FRenderResource {

public:
    FStreamingManager();
    //Initialization/releaseRHIresource. virtualvoid
    InitRHI() override; virtualvoid ReleaseRHI()
    override;
```

```
        //Add and delete resources.
        void        Add( FResources* Resources ); Remove
        void        ( FResources* Resources );

        //Must beNaniteCalled once per frame before any rendering occurs, and must also be called inEndUpdate[Before] call.
        ENGINE_APIvoid BeginAsyncUpdate(FRDGBuilder& GraphBuilder); //Must beNaniteCalled once per
        frame before any rendering occurs, and must also be called inBeginUpdate[After] call.

        ENGINE_APIvoid EndAsyncUpdate(FRDGBuilder& GraphBuilder); ENGINE_API
        bool IsAsyncUpdateInProgress(); //Called once per frame after the last request
        was added.
        ENGINE_APIvoid SubmitFrameStreamingRequests(FRDGBuilder& GraphBuilder);

        (......)

private:
        friendclassFStreamingUpdateTask;

        //Heap buffer, including data and upload buffers.
        structFHeapBuffer
        {
                int32                           TotalUpload =0;

                FGrowOnlySpanAllocator          Allocator;

                FScatterUploadBuffer            UploadBuffer;
                FRWByteAddressBuffer            DataBuffer;

                void        Release()
                {
                        UploadBuffer.Release();
                        DataBuffer.Release();
                }
        };

        // FPackedCluster*, GeometryData { Index, Position, TexCoord, TangentX, TangentZ }* FHeapBuffer
                                        ClusterPageData;
        FHeapBuffer                     ClusterPageHeaders;
        FScatterUploadBuffer            ClusterFixupUploadBuffer;
        FHeapBuffer                         Hierarchy;      // Hierarchy.
        FHeapBuffer                         RootPages;      //  Root page.
        TRefCountPtr<FRDGPooledBuffer>StreamingRequestsBuffer;

        uint32  uint32                  MaxStreamingPages;
        uint32 uint32                   MaxPendingPages;
        //Return data.                  MaxPageInstallsPerUpdate;
        uint32                          MaxStreamingReadbackBuffers;
        uint32

                                        ReadbackBuffersWriteIndex;
                                        ReadbackBuffersNumPending;

        TArray<uint32>                  NextRootPageVersion;
        uint32      uint32              NextUpdateIndex;
        uint32    uint32                NumRegisteredStreamingPages;
                                        NumPendingPages;
                                        NextPendingPageIndex;
```

```
	TArray<FRootPageInfo>		RootPageInfos;

#if	!UE_BUILD_SHIPPING
	uint64				PrevUpdateTick;
#endif

	TArray<	FRHIGPUBufferReadback* >		StreamingRequestReadbackBuffers;
	TArray<	FResources* >				PendingAdds;

	TMap< uint32, FResources* > TMap< FPageKey,		RuntimeResourceMap;
	FStreamingPageInfo* > updated immediately.		RegisteredStreamingPagesMap;		// This
is
	TMap<FPageKey, FStreamingPageInfo*>			CommittedStreamingPageMap;		// This
update is deferred to the point where the page has been loaded and committed to memory.
	TArray< FStreamingRequest >				PrioritizedRequestsHeap;
	FStreamingPageInfo					StreamingPageLRU;

	FStreamingPageInfo*					StreamingPageInfoFreeList;
	TArray< FStreamingPageInfo	>			StreamingPageInfos;
	//The repair information of the resident stream page needs to be kept so that the page can be released.
	TArray< FFixupChunk* >					StreamingPageFixupChunks;

	TArray< FPendingPage	>				PendingPages;
#if	!WITH_EDITOR
	TArray<	uint8	>					PendingPageStagingMemory;
#endif
	TArray<	uint8	>					PendingPageStagingMemoryLZ;

	FRequestsHashTable*					RequestsHashTable = nullptr;
	FStreamingPageUploader*					PageUploader = nullptr;

	FGraphEventArray				AsyncTaskEvents;
	FAsyncState					AsyncState;

	//Operation page.
	void CollectDependencyPages(FResources* Resources, TSet< FPageKey >& DependencyPages, constFPageKey&
	Key );
	void SelectStreamingPages( FResources* Resources, TArray< FPageKey >& SelectedPages, TSet<FPageKey>&
SelectedPagesSet, uint32 RuntimeResourceID, uint32 PageIndex, uint32 MaxSelectedPages );


	//Register/Unregister page .
	void RegisterStreamingPage( FStreamingPageInfo* Page,constFPageKey& Key ); void UnregisterPage(
	constFPageKey& Key ); void MovePageToFreeList( FStreamingPageInfo* Page );


	void ApplyFixups(constFFixupChunk& FixupChunk,constFResources& Resources, uint32 PageIndex, uint32
	GPUPageIndex );

	bool ArePageDependenciesCommitted(uint32 RuntimeResourceID, uint32 PageIndex, uint32
DependencyPageStart, uint32 DependencyPageNum);

	//Returns whether any work has been completed and the page/level buffer has been transitioned to a compute-writable state.
	bool ProcessNewResources(FRDGBuilder& GraphBuilder);

	uint32DetermineReadyPages();
	void InstallReadyPages( uint32 NumReadyPages );
```

```
    //Asynchronous updates.
    void AsyncUpdate();

    void ClearStreamingRequestCount(FRDGBuilder& GraphBuilder, FRDGBufferUAVRef BufferUAVRef);

};

// NaniteStream manager declaration.

externENGINE_API TGlobalResource<FStreamingManager>GStreamingManager; }
```

- **NaniteRender**

```
// Engine\Source\Runtime\Renderer\Private\Nanite\NaniteRender.h

staticconstexpr uint32 NANITE_MAX_MATERIALS =64;
staticconstexpr uint32 MAX_VIEWS_PER_CULL_RASTERIZE_PASS_BITS =12; staticconstexpr
uint32 MAX_VIEWS_PER_CULL_RASTERIZE_PASS_MASK                    = ( (1<<
MAX_VIEWS_PER_CULL_RASTERIZE_PASS_BITS ) -1); staticconstexpr uint32
MAX_VIEWS_PER_CULL_RASTERIZE_PASS MAX_VIEWS_PER_CULL_RASTERIZE_PASS_BITS );= (1<<



(......)

// NaniteUnify buffer parameters.

BEGIN_GLOBAL_SHADER_PARAMETER_STRUCT(FNaniteUniformParameters,          )
    SHADER_PARAMETER(FIntVector4,                                SOAStrides
    SHADER_PARAMETER(FIntVector4,                                MaterialConfig)// .x mode, .yz grid
size, .w unused
    SHADER_PARAMETER(uint32,   SHADER_PARAMETER(uint32,      MaxNodes
    SHADER_PARAMETER(uint32,                                 MaxVisibleClusters)
    SHADER_PARAMETER(FVector4,                        offset RenderFlags)
    SHADER_PARAMETER_SRV(ByteAddressBuffer,                      RectScaleOffset// xy: scale, zw:
    SHADER_PARAMETER_SRV(ByteAddressBuffer,
    SHADER_PARAMETER_SRV(ByteAddressBuffer,                      ClusterPageData)
    SHADER_PARAMETER_SRV(StructuredBuffer<uint>,                 ClusterPageHeaders)
    SHADER_PARAMETER_TEXTURE(Texture2D<uint2>,                   VisibleClustersSWHW)
    SHADER_PARAMETER_TEXTURE(Texture2D<UlongType>,    S     VisibleMaterials
    HADER_PARAMETER_TEXTURE(Texture2D<UlongType>,                MaterialRange)
    SHADER_PARAMETER_TEXTURE(Texture2D<uint>,                    VisBuffer64)
    END_SHADER_PARAMETER_STRUCT()                                DbgBuffer64)
                                                                 DbgBuffer32)



(......)

//Rasterization parameters.

BEGIN_SHADER_PARAMETER_STRUCT( FRasterParameters, )
    SHADER_PARAMETER_RDG_TEXTURE_UAV( RWTexture2D< uint >,                  OutDepthBuffer )//depth
    SHADER_PARAMETER_RDG_TEXTURE_UAV( RWTexture2D< UlongType >,              OutVisBuffer64 )//Visible
sex
    SHADER_PARAMETER_RDG_TEXTURE_UAV( RWTexture2D< UlongType >,              OutDbgBuffer64 )//debug
data
    SHADER_PARAMETER_RDG_TEXTURE_UAV(  RWTexture2D<  uint  >,               OutDbgBuffer32 )
    SHADER_PARAMETER_RDG_TEXTURE_UAV(  RWTexture2D<  uint  >,               LockBuffer )//Lock Buffer
    END_SHADER_PARAMETER_STRUCT()
```

```
// NaniteDraw command information.
classFNaniteCommandInfo {


public:

    staticconstexpr int32 MAX_STATE_BUCKET_ID = (1<<14) -1;// Must match NaniteDataDecode.ush


    void SetStateBucketId(int32 InStateBucketId) {

        StateBucketId = InStateBucketId;
    }

    int32GetStateBucketId() {          const

        check(StateBucketId <          MAX_STATE_BUCKET_ID);
        returnStateBucketId;
    }

    uint32GetMaterialId()const {

        returnGetMaterialId(GetStateBucketId());
    }

    staticuint32GetMaterialId(int32 StateBucketId) {

        floatDepthId = GetDepthId(StateBucketId); return
        *reinterpret_cast<uint32*>(&DepthId);
    }

    static float GetDepthId(int32 StateBucketId) {

        returnfloat(StateBucketId +1) /float(MAX_STATE_BUCKET_ID);
    }

private:
    //Store the index to the correspondingFMeshDrawCommandofFScene::NaniteDrawCommandsmiddle.
    int32 StateBucketId = INDEX_NONE;

};

structMeshDrawCommandKeyFuncs;

// NaniteDraw command list context, follow nonNaniteThe comparison type of the pattern.
classFNaniteDrawListContext: public FMeshPassDrawListContext {


public:
    FNaniteDrawListContext(FRWLock& InNaniteDrawCommandLock, FStateBucketMap&
    InNaniteDrawCommands);
    virtual FMeshDrawCommand&AddCommand(FMeshDrawCommand& Initializer, uint32
NumElements) override final;

    virtualvoid        FinalizeCommand(
        const        FMeshBatch& MeshBatch,
        int32      BatchElementIndex,          int32
        DrawPrimitiveId,                       int32
        ScenePrimitiveId,
        ERasterizerFillMode          MeshFillMode,
        ERasterizerCullMode        MeshCullMode,
```

```
            FMeshDrawCommandSortKey SortKey,
            EFVisibleMeshDrawCommandFlags constFlags,
                    FGraphicsMinimalPipelineStateInitializer&                PipelineState,
            const   FMeshProcessorShaders* ShadersForDebugging,
            FMeshDrawCommand&  MeshDrawCommand
            ) override final;

        (……)

private:
        FRWLock* NaniteDrawCommandLock;
        FStateBucketMap* NaniteDrawCommands;// NaniteDrawing commands.
        FNaniteCommandInfo CommandInfo;// NaniteCommand information.
        FMeshDrawCommand MeshDrawCommandForStateBucketing;

};

// NaniteColorizer parent class.

classFNaniteShader: public FGlobalShader {

public:
        (……)

        static bool ShouldCompilePermutation(constFGlobalShaderPermutationParameters& Parameters);

        static void ModifyCompilationEnvironment(constFGlobalShaderPermutationParameters& Parameters,
FShaderCompilerEnvironment& OutEnvironment;
};

//Specifies a full-screen vertex shader for depth drawing, available on all platforms.

class FNaniteMaterialVS : public FNaniteShader {

        DECLARE_GLOBAL_SHADER(FNaniteMaterialVS);

        BEGIN_SHADER_PARAMETER_STRUCT(FParameters,          )
                SHADER_PARAMETER(float, MaterialDepth)
                END_SHADER_PARAMETER_STRUCT()

        (……)

        voidGetShaderBindings(
                constFScene* Scene, ERHIFeatureLevel::Type FeatureLevel, const
                FPrimitiveSceneProxy* PrimitiveSceneProxy,const FMaterialRenderProxy&
                                                MaterialRenderProxy,                const
                FMaterial& Material,constFMeshPassProcessorRenderState&
                DrawRenderState,          const       FMeshMaterialShaderElementData&
                                                ShFMadeesrhEDlermawenSitnDgalteaS,haderBindings&
                ShaderBindingsconst

        {
                ShaderBindings.Add(NaniteUniformBuffer,                DrawRenderState.GetNaniteUniformBuffer());
        }

private:
        LAYOUT_FIELD(FShaderParameter, MaterialDepth);
        LAYOUT_FIELD(FShaderUniformBufferParameter, NaniteUniformBuffer);
};

// NaniteGrid processor.
```

```
class FNaniteMeshProcessor : public FMeshPassProcessor {

public:
	(……)

	virtualvoidAddMeshBatch(constFMeshBatch& RESTRICT MeshBatch, uint64 BatchElementMask,const
FPrimitiveSceneProxy* RESTRICT PrimitiveSceneProxy, int32 StaticMeshId =-1) override final;


private:
	FMeshPassProcessorRenderState		PassDrawRenderState;
};

//createNaniteGrid processor instance.
FMeshPassProcessor* CreateNaniteMeshProcessor(constFScene* Scene,constFSceneView*
InViewIfDynamicMeshCommand, FMeshPassDrawListContext* InDrawListContext);

// NaniteMaterial table.
class FNaniteMaterialTables {

public:
	FNaniteMaterialTables(uint32 MaxMaterials = NANITE_MAX_MATERIALS);
	~FNaniteMaterialTables();

	voidRelease();

	voidUpdateBufferState(FRDGBuilder& GraphBuilder, uint32 NumPrimitives);

	voidBegin(FRHICommandListImmediate& RHICmdList, uint32 NumPrimitives, uint32
	NumPrimitiveUpdates);
	void* GetDepthTablePtr(uint32 PrimitiveIndex, uint32 EntryCount); void
	Finish(FRHICommandListImmediate& RHICmdList);

	FRHIShaderResourceView* GetDepthTableSRV()const{returnDepthTableDataBuffer.SRV; }

private:
	uint32 MaxMaterials =0; uint32
	NumPrimitiveUpdates =0; uint32
	NumDepthTableUpdates =0; uint32
	NumHitProxyTableUpdates =0;
	//CPUAnd data buffer for upload.

	FScatterUploadBuffer		DepthTableUploadBuffer;
	FRWByteAddressBuffer		DepthTableDataBuffer;
	FScatterUploadBuffer		HitProxyTableUploadBuffer;
	FRWByteAddressBuffer		HitProxyTableDataBuffer;
};

namespace Nanite
{
//Rasterization technology.
enumclass ERasterTechnique : uint8 {


	LockBufferFallback =0,//		Use a standby lock buffer to approximate no64Bit atomicity (with race conditions	) .
	PlatformAtomics =1,//Use the platform provided6Bit4Atom .
	NVAtomics =2,//UsageNExpandvThe booths provided6Origina4lSon
	AMDAtomicsD3D11=3,//UsageAMD AMDAExttoenmsioicns(D3D11)2=4,//		Provided64BPoistitAiontoSomn..
	UsageAMDExtensions(D3D12)Provided
```

```
        DepthOnly =5,//Use for depth32Atomic, no extra overhead.


        NumTechniques
};

//Rasterization scheduling mode.
enumclass ERasterScheduling : uint8 {

        HardwareOnly =0,//Rasterization using only fixed-function hardware.
        HardwareThenSoftware =1,//Rasterize the large triangle using hardware, and use software (comtue shader)Rasterize small triangles.
        HardwareAndSoftwareOverlap =2,//Rasterize the large triangle using hardware, and then rasterize it using software (comtue shader)Rasterize small triangles.

};

//Output buffer mode. Used to select the rasterization mode when creating the device context.
enumclass EOutputBufferMode : uint8 {

        VisBuffer,//     Visibility buffer, the default mode is used to output     IDand depth.
        DepthOnly,//Rasterize depth only up to3Bit 2Buffer .
};
//The filled view.

structFPackedView
{
        FMatrix                 TranslatedWorldToView;
        FMatrix                 TranslatedWorldToClip;
        FMatrix                 ViewToClip;
        FMatrix                 ClipToWorld;

        FMatrix                 PrevTranslatedWorldToView;
        FMatrix                 PrevTranslatedWorldToClip;
        FMatrix                 PrevViewToClip;
        FMatrix                 PrevClipToWorld;

        FIntVector4             ViewRect;
        FVector4        ViewSizeAndInvSize;
        FVector4        ClipSpaceScaleOffset;
        FVector4        PreViewTranslation;
        FVector4        PrevPreViewTranslation;
        FVector4        WorldCameraOrigin;
        FVector4        ViewForwardAndNearPlane;

        FVector2D           LODScales;
        float               MinBoundsRadiusSq;
        uint32                StreamingPriorityCategory_AndFlags;

        FIntVector4     TargetLayerIdX_AndMipLevelY_AndNumMipLevelsZ;
        FIntVector4             HZBTestViewRect;            // In full resolution

        //calculateLODFor example, assuming the view size and projection are already set. Depends on global variables

        GNaniteMaxPixelsPerEdge. voidUpdateLODScales();
};

//Crop context.
structFCullingContext
{
        FGlobalShaderMap* ShaderMap;
```

```
        uint32          uint32          Duri antPa3s2sIndueixn;t32
        TRefCountPtr<IPoolNdRe udmerITnasrtgaentc>esPreCull;
                                        RenderFlags;
                                        DebugFlags;
                                                                PrevHZB;//If notnull, HZBCropping will start.
        FIntRect                    HZBBuildViewRect;
        bool bool                   bTwoPassOcclusion;
                                    bSupportsMultiplePasses;

        FIntVector4                     SOAStrides;

        FRDGBufferRef               MainRasterizeArgsSWHW;
        FRDGBufferRef               PostRasterizeArgsSWHW;

        FRDGBufferRef               SafeMainRasterizeArgsSWHW;
        FRDGBufferRef               SafePostRasterizeArgsSWHW;

        FRDGBufferRef               MainAndPostPassPersistentStates;
        FRDGBufferRef               VisibleClustersSWHW;
        FRDGBufferRef               OccludedInstances;
        FRDGBufferRef               OccludedInstancesArgs;
        FRDGBufferRef               TotalPrevDrawClustersBuffer;
        FRDGBufferRef               StreamingRequests;
        FRDGBufferRef               ViewsBuffer;
        FRDGBufferRef               InstanceDrawsBuffer;
        FRDGBufferRef               StatsBuffer;

};
//Rasterize context.

structFRasterContext
{
        FGlobalShaderMap*           ShaderMap;

        FVector2D                       RcpViewSize;
        FIntPoint                       TextureSize;
        ERasterTechnique            RasterTechnique;
        ERasterScheduling               RasterScheduling;

        FRasterParameters               Parameters;

        FRDGTextureRef                  LockBuffer;
        FRDGTextureRef                  DepthBuffer;
        FRDGTextureRef                  VisBuffer64;
        FRDGTextureRef                  DbgBuffer64;
        FRDGTextureRef                  DbgBuffer32;

        uint32                          VisualizeModeBitMask;
        bool                        VisualizeActive;
};

//Rasterization result.

structFRasterResults
{
        FIntVector4                     SOAStrides;
        uint32                      MaxVisibleClusters;
        uint32                      MaxNodes;
        uint32                      RenderFlags;
```

```
        FRDGBufferRef           ViewsBuffer{};
        FRDGBufferRef           VisibleClustersSWHW{};

        FRDGTextureRef          VisBuffer64{};
        FRDGTextureRef          DbgBuffer64{};
        FRDGTextureRef          DbgBuffer32{};

        FRDGTextureRef          MaterialDepth{};
        FRDGTextureRef          NaniteMask{};
        FRDGTextureRef          VelocityBuffer{};

        TArray<FVisualizeResult, TInlineAllocator<32>> Visualizations;
};

//Initialize the clipping context.

FCullingContext                 InitCullingContext(FRDGBuilder& GraphBuilder,constFScene& Scene,
. . . ) ;
//Initialize the rasterization context.
FRasterContext InitRasterContext(FRDGBuilder& GraphBuilder, ERHIFeatureLevel::Type FeatureLevel, ...);



//The populated view parameters.

structFPackedViewParams {

        FViewMatrices       ViewMatrices;
        FViewMatrices       PrevViewMatrices;
        FIntRect        ViewRect;
        FIntPoint       RasterContextSize;
        uint32 StreamingPriorityCategory =0; float
        MinBoundsRadius =0.0f; floatLODScaleFactor =1.0f
        ; uint32 Flags =0;


        int32       TargetLayerIndex            =       0;      int32
        PrevTargetLayerIndex = INDEX_NONE; int32
        TargetMipLevel =0; int32 TargetMipCount =1;


        FIntRect HZBTestViewRect = {0,0,0,0};
};

FPackedView CreatePackedView(constFPackedViewParams& Params ); FPackedView
CreatePackedViewFromViewInfo(constFViewInfo& View, FIntPoint RasterContextSize, ...);


//Rasterization status.

structFRasterState
{
        boolbNearClip =true;//Is it enabled?NearFlat cutting.
        ERasterizerCullMode CullMode = CM_CW;//Rasterizer clipping mode, default is clockwise.

};

//Rasterization with clipping.

voidCullRasterize
        FRDGBuilder& GraphBuilder, const
        FScene& Scene,
        constTArray<FPackedView, SceneRenderingAllocator>& Views,
        FCullingContext& CullingContext,
```

```
        constFRasterContext& RasterContext,
        constFRasterState& RasterState = FRasterState(),
        constTArray<FInstanceDraw, SceneRenderingAllocator>* OptionalInstanceDraws = nullptr, boolbExtractStats =false

) ;

//    Rasterize to virtual shadow map (virtual shadow map)set
voidCullRasterize
        FRDGBuilder& GraphBuilder, const
        FScene& Scene,
        constTArray<FPackedView, SceneRenderingAllocator>& Views, uint32
        NumPrimaryViews,                    // Number of non-mip views
        FCullingContext& CullingContext, constFRasterContext& RasterContext,

        constFRasterState& RasterState = FRasterState(),
        constTArray<FInstanceDraw, SceneRenderingAllocator>* OptionalInstanceDraws = nullptr,
        FVirtualShadowMapArray* VirtualShadowMapArray = nullptr,
        boolbExtractStats =false
) ;

//Decompress the rasterization result.

voidExtractResults(FRDGBuilder& GraphBuilder,constFCullingContext& CullingContext, constFRasterContext&
RasterContext, FRasterResults& RasterResults);

//Triggers the shadow map.

voidEmitShadowMap(FRDGBuilder& GraphBuilder,constFRasterContext& RasterContext,const FRDGTextureRef
DepthBuffer, ...);
//Trigger cube shadows.
voidEmitCubemapShadow(FRDGBuilder& GraphBuilder,constFRasterContext& RasterContext, constFRDGTextureRef
CubemapDepthBuffer, ...);



//Trigger depth target.

voidEmitDepthTargets(FRDGBuilder& GraphBuilder,constFScene& Scene,constFViewInfo& View, ...);



//drawBasePass.
voidDrawBasePass(FRDGBuilder& GraphBuilder,constFSceneTextures& SceneTextures,const FDBufferTextures&
DBufferTextures,constFScene& Scene,constFViewInfo& View,const FRasterResults& RasterResults );



//drawLumenGrid capture channel.

voidDrawLumenMeshCapturePass(FRDGBuilder& GraphBuilder,constFScene& Scene, ...);

(.....)
}

//Is rendering required?Nanite.

externboolShouldRenderNanite(constFScene* Scene,constFViewInfo& View,bool bCheckForAtomicSupport
=true);
```

- **NaniteSceneProxy**

```
// Engine\Source\Runtime\Engine\Public\NaniteSceneProxy.h

namespace Nanite
```

```
{

// NaniteScene proxy parent class.
classFSceneProxyBase: public FPrimitiveSceneProxy {


public:
    structFMaterialSection {

        UMaterialInterface* Material = nullptr; int32
        MaterialIndex = INDEX_NONE;
    };

public:
    ENGINE_API SIZE_TGetTypeHash()constoverride;

    FSceneProxyBase(UPrimitiveComponent* :          Component)
      FPrimitiveSceneProxy(Component)
    {
        bIsNaniteMesh      =   true;
        bAlwaysVisible      =  true;
    }

    //   Check whether it is satisfiedNanite  Rendering conditions: no opaque objects, no decals,          Masked,    Not normal translucency, not separated translucency.
    static bool IsNaniteRenderable(FMaterialRelevance MaterialRelevance) {

        returnMaterialRelevance.bOpaque &&
        !MaterialRelevance.bDecal !                                  &&
        MaterialRelevance.bMasked !                                  &&
        MaterialRelevance.bNormalTranslucency !                      &&
        MaterialRelevance.bSeparateTranslucency;
    }

    virtualbool CanBeOccluded()constoverride;
    inlineconstTArray<FMaterialSection>&GetMaterialSections()const; inlineint32
    GetMaterialMaxIndex()const;
    virtualconstTArray<FPrimitiveInstance>*GetPrimitiveInstances()const; virtual
    TArray<FPrimitiveInstance>*GetPrimitiveInstances(); virtual uint8
    GetCurrentFirstLODIdx_RenderThread()constoverride;


protected:
    ENGINE_APIvoid DrawStaticElementsInternal(FStaticPrimitiveDrawInterface* PDI,const FLightCacheInterface*
    LCI);

protected:
    TArray<FMaterialSection> MaterialSections;
    TArray<FPrimitiveInstance> Instances; int32
    MaterialMaxIndex = INDEX_NONE;
};

// NaniteScene Agent.
classFSceneProxy: public FSceneProxyBase {

public:
    FSceneProxy(UStaticMeshComponent* Component);
    FSceneProxy(UInstancedStaticMeshComponent* Component);
    FSceneProxy(UHierarchicalInstancedStaticMeshComponent* virtual          Component);
    ~FSceneProxy() =default;
```

```
public:
	// FPrimitiveSceneProxyInterface.
	virtual FPrimitiveViewRelevance GetViewRelevance(constFSceneView* View)const override;
	virtualvoid GetLightRelevance(constFLightSceneProxy* LightSceneProxy,bool& bDynamic,bool& bRelevant, bool& bLightMapped,bool& bShadowMapped)constoverride;

//	Gets a static or dynamic grid element.
virtualvoid DrawStaticElements(FStaticPrimitiveDrawInterface* PDI) override; virtualvoid GetDynamicMeshElements(constTArray<constFSceneView*>& Views,const FSceneViewFamily& ViewFamily, uint32 VisibilityMap, FMeshElementCollector& Collector) constoverride;


	//Ray tracing related interfaces.
#if RHI_RAYTRACING
	virtualbool IsRayTracingRelevant()const{returntrue; } virtualbool IsRayTracingStaticRelevant()const{returnfalse; }
	virtualvoid GetDynamicRayTracingInstances(FRayTracingMaterialGatheringContext& Context, TArray<structFRayTracingInstance>& OutRayTracingInstances) override;
#endif

	virtual uint32GetMemoryFootprint()constoverride;

	virtualvoid GetLCIs(FLCIArray& LCIs) override {

		FLightCacheInterface* LCI = &MeshInfo;
		LCIs.Add(LCI);
	}

	//Distance Field Interface.
	virtualvoid GetDistancefieldAtlasData(constFDistanceFieldVolumeData*& OutDistanceFieldData, float& SelfShadowBias)constoverride;
	virtualvoid GetDistancefieldInstanceData(TArray<FMatrix>& ObjectLocalToWorldTransforms)constoverride;
	virtualbool HasDistanceFieldRepresentation()constoverride;
	//GIInterface.
	virtualconstFCardRepresentationData*GetMeshCardRepresentation()constoverride; virtual int32 GetLightMapCoordinateIndex()constoverride;


	//Get a static grid.
	constUStaticMesh*GetStaticMesh()const {

		returnStaticMesh;
	}

protected:
	virtualvoid CreateRenderThreadResources() override;

	classFMeshInfo: public FLightCacheInterface {

	public:
		FMeshInfo(constUStaticMeshComponent* InComponent);

		// FLightCacheInterface.
		virtual FLightInteractionGetInteraction(constFLightSceneProxy* LightSceneProxy) override;
const
```

```
        private:
            TArray<FGuid>       IrrelevantLights;
        };

        bool IsCollisionView(constFEngineShowFlags& EngineShowFlags,bool&
bDrawSimpleCollision,bool& bDrawComplexCollision)const;

protected:
        FMeshInfo MeshInfo;

        FResources* Resources = nullptr;

        const    FStaticMeshRenderData* RenderData;
        const    FDistanceFieldVolumeData*          DistanceFieldData;
        const    FCardRepresentationData*         CardRepresentationData;

        FMaterialRelevance MaterialRelevance;

        uint32 bReverseCulling :1; uint32
        bHasMaterialErrors :1;

        constUStaticMesh* StaticMesh = nullptr;

#if    RHI_RAYTRACING
        TArray<FRayTracingGeometry*>          RayTracingGeometries;
#endif

        (......)
};

}// namespace Nanite
```

- **RenderUtils**

```
// Engine\Source\Runtime\RenderCore\Public\RenderUtils.h

(......)

//Check whether the platform supportsNaniteRendering.
RENDERCORE_APIbool DoesPlatformSupportNanite(EShaderPlatform Platform) {

        //Make sure the current platform definesDDPI(FGenericDataDrivenShaderPlatformInfo).
        const boolbValidPlatform = FDataDrivenShaderPlatformInfo::IsValid(Platform); // NaniteneedGPUScene.
        const boolbSupportGPUScene =


FDataDrivenShaderPlatformInfo::GetSupportsGPUScene(Platform);
        // NaniteSpecific testing.
        const boolbSupportNanite=
FDataDrivenShaderPlatformInfo::GetSupportsNanite(Platform);

        const boolbFullCheck = bValidPlatform && bSupportGPUScene && bSupportNanite; returnbFullCheck;

}

//UsageNanite,If successful, it will returntrue.
inlinebool UseNanite(EShaderPlatform ShaderPlatform,boolbCheckForAtomicSupport =true); //UsageVSM,Successful
returntrue.
```

```
inlinebool UseVirtualShadowMaps(EShaderPlatform ShaderPlatform,constFStaticFeatureLevel FeatureLevel);

//Use nonNaniteofVSM,Successful returntrue.The premise isr.Shadow.Virtual.NonNaniteVSMNot for0,and
UseVirtualShadowMapsfortrue.
inlinebool UseNonNaniteVirtualShadowMaps(EShaderPlatform ShaderPlatform,const FStaticFeatureLevel
FeatureLevel);
```

- **other**

```
// Engine\Source\Runtime\Engine\Classes\Components\StaticMeshComponent.h

classENGINE_API UStaticMeshComponent: public UMeshComponent {

    (......)

    uint8 bDisplayNaniteProxyMesh:1;//forNaniteEnabled grid, iftrue, only the proxy grid will be displayed.


    (......)
};

// Engine\Source\Runtime\Engine\Public\PrimitiveSceneProxy.h

classFPrimitiveSceneProxy {

    inlinebool IsNaniteMesh()const {

        returnbIsNaniteMesh;
    }

    (......)

private:
    uint8 bIsNaniteMesh :1;//whetherNaniteGrid.


    (......)
};

//If the specified grid can beNaniteRendering, then returntrue.

ENGINE_APIexternbool SupportsNaniteRendering(constFVertexFactory* RESTRICT VertexFactory,const
FPrimitiveSceneProxy* RESTRICT PrimitiveSceneProxy); ENGINE_APIexternbool SupportsNaniteRendering(const
FVertexFactory* RESTRICT VertexFactory,constFPrimitiveSceneProxy* RESTRICT PrimitiveSceneProxy,constclass
FMaterialRenderProxy* MaterialRenderProxy, ERHIFeatureLevel::Type FeatureLevel);



// Engine\Source\Runtime\Renderer\Public\MeshPassProcessor.h

structFMeshPassProcessorRenderState {

public:
    void SetNaniteUniformBuffer(FRHIUniformBuffer* InNaniteUniformBuffer);
    FRHIUniformBuffer*GetNaniteUniformBuffer()const;

    (......)

private:
    FRHIUniformBuffer* NaniteUniformBuffer = nullptr;// NaniteUnified buffer.
```

```
    (.....)
};

// Engine\Source\Runtime\RenderCore\Public\VertexFactory.h

//Vertex Factory marker.
enum classEVertexFactoryFlags: uint32 {

    N=on0eu, =1u<<1, =1u<<2, =1u<<3, = U
    1usedWit<hM<ateri4a,ls =1u<<5,
    SupportsCtacahtincgLMieghsthDirngawCommands =1u<<6,
    SupportsDynamicLighting
    SupportsPrecisePrevWorldPos
    SupportsPositionOnly

    SupportsPrimitiveIdStream                =1u<<7,
    SupportsNaniteRendering                  =1u<<8,//SupportNaniteRendering.
};
//SupportNaniteRendering.

bool SupportsNaniteRendering()const{return
HasFlags(EVertexFactoryFlags::SupportsNaniteRendering);                    }

// Engine\Source\Runtime\RHI\Public\RHIDefinitions.h

//General data-driven shader platform information.
classRHI_API FGenericDataDrivenShaderPlatformInfo {

    staticFORCEINLINE_DEBUGGABLEconst bool GetSupportsNanite(constFStaticShaderPlatform Platform)

    {
        returnInfos[Platform].bSupportsNanite;
    }

    (.....)
};
```

6.4.3.3 Nanite rendering process

The main rendering steps of Nanite also occur in the following

FDeferredShadingSceneRenderer::Render. The following will explain the steps related to Nanite and the important steps involved in the previous articles:

```
void FDeferredShadingSceneRenderer::Render(FRDGBuilder& GraphBuilder) {

    //Try usingNaniteRendering.
    const boolbNaniteEnabled = UseNanite(ShaderPlatform) &&
ViewFamily.EngineShowFlags.NaniteMeshes;

    //Update primitive scene information.
    Scene->UpdateAllPrimitiveSceneInfos(GraphBuilder,true);

    //UsageGPUScene.
    FGPUSceneScopeBeginEndHelper        GPUSceneScopeBeginEndHelper(Scene->GPUScene,
GPUSceneDynamicContext, Scene);
```

```
boolbVisualizeNanite =false; if(bNaniteEnabled)//
NaniteOnly executed when enabled {

    //renewNaniteGlobal resources. Need to beNaniteManage out-of-
    order buffers. Nanite::GGlobalResources.Update(GraphBuilder);
    //Start asynchronous updateNaniteStream Manager.
    Nanite::GStreamingManager.BeginAsyncUpdate(GraphBuilder);

    //deal withNaniteVisualization mode.
    FNaniteVisualizationData& NaniteVisualization = GetNaniteVisualizationData(); if(Views.Num() >0) {

        constFName& NaniteViewMode = Views[0].CurrentNaniteVisualizationMode; if
        (NaniteVisualization.Update(NaniteViewMode)) {

            ViewFamily.EngineShowFlags.SetVisualizeNanite(true);
        }
        b Visualize Nanite                                    =       N anite V isualizati                                    on
ViewFamily.EngineShowFlags.VisualizeNanite;
    }
}

(.....)

//Is application required?NaniteMaterial.
const     boolbShouldApplyNaniteMaterials
      =  !ViewFamily.EngineShowFlags.ShaderComplexity !
    && ViewFamily.UseDebugViewPS()
    && !ViewFamily.EngineShowFlags.Wireframe !
    && ViewFamily.EngineShowFlags.LightMapDensity;

(.....)

//Instantiate the clipping manager.
FInstanceCullingManagerInstanceCullingManager(GInstanceCullingManagerResources,
Scene->GPUScene.IsEnabled());

bDoInitViewAftersPrepass = InitViews(GraphBuilder, ..., InstanceCullingManager);

(.....)
//deal withGPUScene.
{

    (.....)

    //renewGPUScene.
    Scene->GPUScene.Update(GraphBuilder, *Scene);

    (.....)

    //Upload dynamic primitive shader data toGPU.
    for(int32 ViewIndex =0; ViewIndex < Views.Num(); ViewIndex++) {

        FViewInfo& View = Views[ViewIndex]; Scene-
        >GPUScene.UploadDynamicPrimitiveShaderDataForView(GraphBuilder,                    Scene,
View);
    }
```

```
        //Instantiated clipping.
        {
            InstanceCullingManager.CullInstances(GraphBuilder, Scene->GPUScene);
        }

        (......)
    }

    (......)

    if(bNaniteEnabled)
    {
        Nanite::ListStatFilters(this);

        //Must be in each frameNaniteCalled before rendering.
        Nanite::GStreamingManager.EndAsyncUpdate(GraphBuilder);

    }

    (......)

    //Advance depth channel.
    RenderPrePass(GraphBuilder, SceneTextures.Depth.Target, InstanceCullingManager);


    (......)

    // NaniteRasterization
    TArray<Nanite::FRasterResults, TInlineAllocator<2>> if                    NaniteRasterResults;
    (bNaniteEnabled && Views.Num() >0) {

        LLM_SCOPE_BYTAG(Nanite);

        NaniteRasterResults.AddDefaulted(Views.Num());

        RDG_GPU_STAT_SCOPE(GraphBuilder, NaniteRaster);
        constFIntPoint RasterTextureSize = SceneTextures.Depth.Target->Desc.Extent;

        constFViewInfo& PrimaryViewRef = Views[0];
        constFIntRect PrimaryViewRect = PrimaryViewRef.ViewRect;

        //Main Rasterized View
        {
            Nanite::FRasterState RasterState;

            Nanite::FRasterContext RasterContext = Nanite::InitRasterContext(GraphBuilder,
FeatureLevel, RasterTextureSize);

            const boolbTwoPassOcclusion =true; const bool
            bUpdateStreaming =true; const bool
            bSupportsMultiplePasses =false;
            const boolbForceHWRaster = RasterContext.RasterScheduling ==
Nanite::ERasterScheduling::HardwareOnly;
            const boolbPrimaryContext =true; const bool
            bDiscardNonMoving =
ViewFamily.EngineShowFlags.DrawOnlyVSMInvalidatingGeo !=0;

            //Traverse allview
            for(int32 ViewIndex =0; ViewIndex < Views.Num(); ViewIndex++) {
```

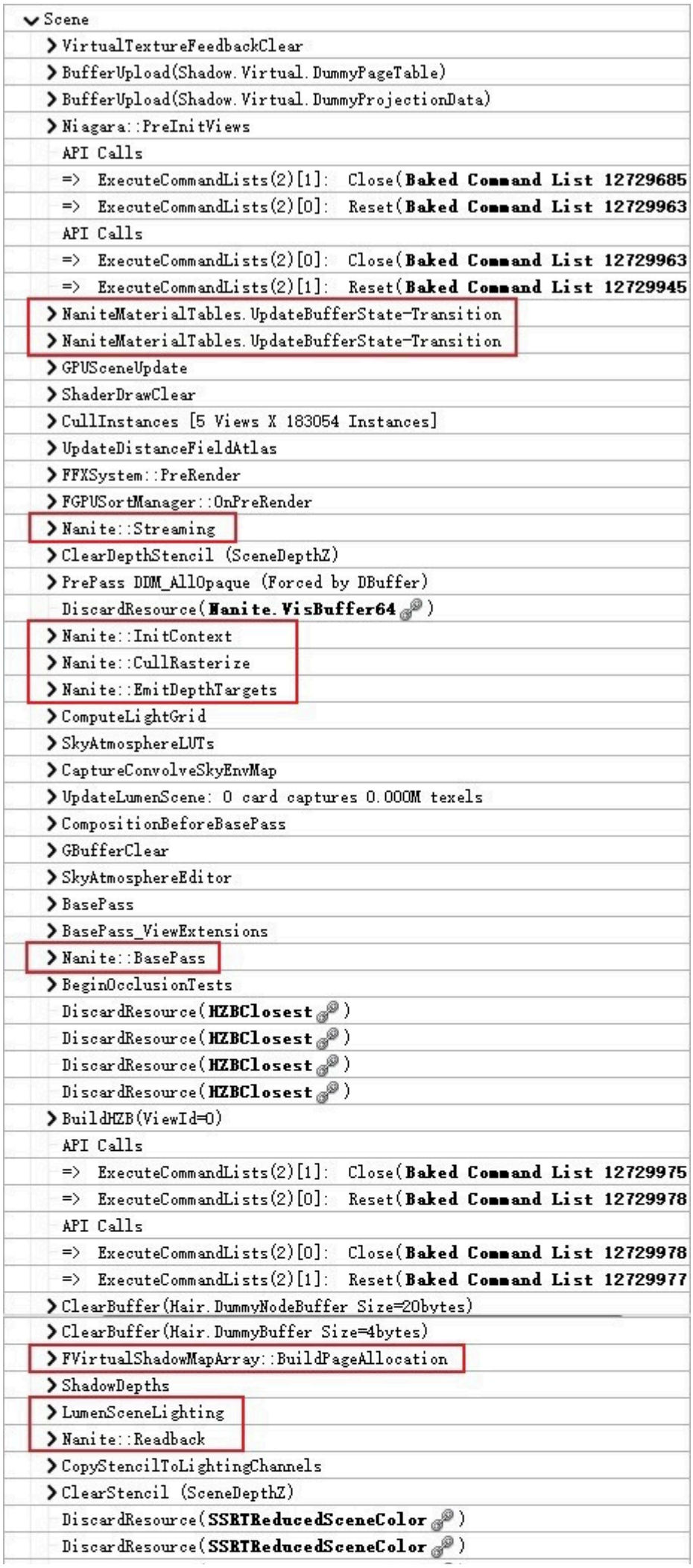
```
const FViewInfo& View = Views[ViewIndex];

//Initialize the clipping context.
Nanite::FCullingContext CullingContext = Nanite::InitCullingContext(
    GraphBuilder,
    * Scene,
    !bIsEarlyDepthComplete ? View.PrevViewInfo.NaniteHZB :
View.PrevViewInfo.HZB,
    View.ViewRect,
    bTwoPassOcclusion,
    bUpdateStreaming,
    bSupportsMultiplePasses,
    bForceHWRaster,
    bPrimaryContext,
    bDiscardNonMoving
);

static FString EmptyFilterName = TEXT("");// Empty filter represents
primary    view.
const bool bExtractStats = Nanite::IsStatFilterActive(EmptyFilterName);

Nanite::FPackedView PackedView =
Nanite::CreatePackedViewFromViewInfo(View, RasterTextureSize, VIEW_FLAG_HZBTEST, /
*StreamingPriorityCategory*/3);

//Rasterization with clipping.
Nanite::CullRasterize(
    GraphBuilder, *Scene, { PackedView },
    CullingContext, RasterContext, RasterState,

    /*OptionalInstanceDraws*/nullptr,
    bExtractStats



);

Nanite::FRasterResults& RasterResults = NaniteRasterResults[ViewIndex];

//  If depth is required in advance, render it.
if  (bNeedsPrePass)
{
    Nanite::EmitDepthTargets(
        GraphBuilder,                    * Scene,
        Views[ViewIndex],
        CullingContext.SOAStrides,
        CullingContext.VisibleClustersSWHW,
        CullingContext.ViewsBuffer,
        SceneTextures.Depth.Target,
        RasterContext.VisBuffer64,
        RasterResults.MaterialDepth,
        RasterResults.NaniteMask,
        RasterResults.VelocityBuffer,
        bNeedsPrePass


    );
}
```

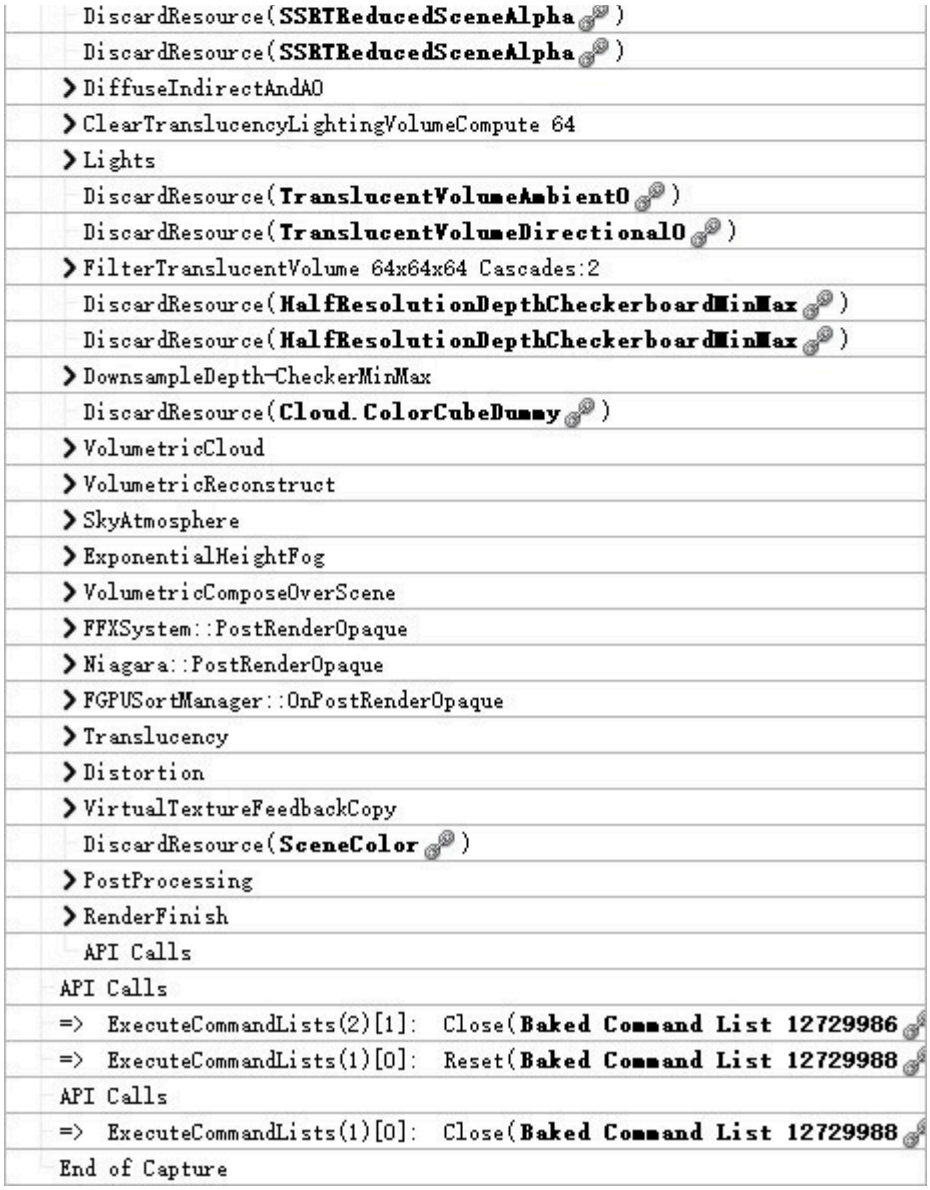
```
                    //Building a hierarchical depth bufferHZB.
                    if(!bIsEarlyDepthComplete && bTwoPassOcclusion && View.ViewState) {

                        //There will not be a full scene depth for the back pass, so the fullHkDJmain channel, otherwise it will interfere
HkDJDestroys occlusion culling.
Back Channel
                        RDG_EVENT_SCOPE(GraphBuilder,"Nanite::BuildHZB");

                        FRDGTextureRef SceneDepth = SystemTextures.Black;
                        FRDGTextureRef GraphHZB = nullptr;

                        //Build to the maximumHZB.
                        BuildHZBFurthest(
                        GraphBuilder,
                        SceneDepth,
                        RasterContext.VisBuffer64,
                        PrimaryViewRect,
                        FeatureLevel,
                        ShaderPlatform,
                        TEXT("Nanite.HZB"),
                        /* OutFurthestHZBTexture = */&GraphHZB );


                        GraphBuilder.QueueTextureExtraction( GraphHZB, &View.ViewState-
> PrevFrameViewInfo.NaniteHZB );
                    }

                    Nanite::ExtractResults(GraphBuilder, CullingContext, RasterContext,
RasterResults);
                }
            }
        }

        (.....)
        //RenderingNaniteofBasePass. {


            RenderBasePass(GraphBuilder, SceneTextures, DBufferTextures, BasePassDepthStencilAccess,
ForwardScreenSpaceShadowMaskTexture, InstanceCullingManager);
            AddServiceLocalQueuePass(GraphBuilder);

            if(bNaniteEnabled && bShouldApplyNaniteMaterials) {

                for(int32 ViewIndex =0; ViewIndex < Views.Num(); ++ViewIndex) {

                    constFViewInfo& View = Views[ViewIndex];
                    Nanite::FRasterResults& RasterResults = NaniteRasterResults[ViewIndex];

                    //  If depth was not drawn earlier, draw it now
                    if  (!bNeedsPrePass)
                    {
                        Nanite::EmitDepthTargets(
                            GraphBuilder,
                            * Scene,
                            Views[ViewIndex],
                            RasterResults.SOAStrides,
                            RasterResults.VisibleClustersSWHW,
                            RasterResults.ViewsBuffer,
                            SceneTextures.Depth.Target,
                            RasterResults.VisBuffer64,
```

```
                                RasterResults.MaterialDepth,
                                RasterResults.NaniteMask,
                                RasterResults.VelocityBuffer,
                                bNeedsPrePass
                        ) ;
                }

                //drawBasePass.
                Nanite::DrawBasePass(
                GraphBuilder,
                SceneTextures,
                DBufferTextures,
                * Scene,
                View,
                RasterResults
                ) ;

            }
        }

        if(!bAllowReadOnlyDepthBasePass) {

            AddResolveSceneDepthPass(GraphBuilder, Views, SceneTextures.Depth);
        }

        (......)
    }

    (......)

    if(bNaniteEnabled)
    {
        //Calculates volumetric fog.
        if(!bOcclusionBeforeBasePass) {


            ComputeVolumetricFog(GraphBuilder);
        }

        //Submit a frame stream request.
        Nanite::GStreamingManager.SubmitFrameStreamingRequests(GraphBuilder);
    }

    (......)

    //Rendering deferred lights.
    RenderLights(GraphBuilder, SceneTextures, ...);


    (......)

    //Rendering semi-transparent objects.
    RenderTranslucency(GraphBuilder, SceneTextures, ...);

    (......)

    //Post-processing
    AddPostProcessingPasses(GraphBuilder, View, PostProcessingInputs, NaniteResults,
InstanceCullingManager);
```

```
        (……)
}
```

It can be seen that the rendering process of Nanite is similar to that of the normal mode. First, the metadata, GPUScene, and clipping data are updated, then BasePass and Lighting are rendered, and finally semi-transparency and post-processing are performed . However, there are also differences from the normal mode, such as the addition of GStreamingManager, FInstanceCullingManager, HZB construction, Nanite rasterization, and other stages. The following example project AncientGame is intercepted with the help of RenderDoc to show the main steps related to UE5:


Scene
VirtualTextureFeedbackClear
BufferUpload(Shadow.Virtual.DummyPageTable)
BufferUpload(Shadow.Virtual.DummyProjectionData)
Niagara::PreInitViews
API Calls
=> ExecuteCommandLists(2)[1]: Close(Baked Command List 12729685
=> ExecuteCommandLists(2)[0]: Reset(Baked Command List 12729963
API Calls
=> ExecuteCommandLists(2)[0]: Close(Baked Command List 12729963
=> ExecuteCommandLists(2)[1]: Reset(Baked Command List 12729945
NaniteMaterialTables.UpdateBufferState-Transition
NaniteMaterialTables.UpdateBufferState-Transition
GPUSceneUpdate
ShaderDrawClear
CullInstances [5 Views X 183054 Instances]
UpdateDistanceFieldAtlas
FFXSystem::PreRender
FGPUSortManager::OnPreRender
Nanite::Streaming
ClearDepthStencil (SceneDepthZ)
PrePass DDM_AllOpaque (Forced by DBuffer)
DiscardResource(Nanite.VisBuffer64)
Nanite::InitContext
Nanite::CullRasterize
Nanite::EmitDepthTargets
ComputeLightGrid
SkyAtmosphereLUTs
CaptureConvolveSkyEnvMap
UpdateLumenScene: 0 card captures 0.000M texels
CompositionBeforeBasePass
GBufferClear
SkyAtmosphereEditor
BasePass
BasePass_ViewExtensions
Nanite::BasePass
BeginOcclusionTests
DiscardResource(HZBClosest)
DiscardResource(HZBClosest)
DiscardResource(HZBClosest)
DiscardResource(HZBClosest)
BuildHZB(ViewId=0)
API Calls
=> ExecuteCommandLists(2)[1]: Close(Baked Command List 12729975
=> ExecuteCommandLists(2)[0]: Reset(Baked Command List 12729978
API Calls
=> ExecuteCommandLists(2)[0]: Close(Baked Command List 12729978
=> ExecuteCommandLists(2)[1]: Reset(Baked Command List 12729977
ClearBuffer(Hair.DummyNodeBuffer Size=20bytes)
ClearBuffer(Hair.DummyBuffer Size=4bytes)
FVirtualShadowMapArray::BuildPageAllocation
ShadowDepths
LumenSceneLighting
Nanite::Readback
CopyStencilToLightingChannels
ClearStencil (SceneDepthZ)
DiscardResource(SSRTReducedSceneColor)
DiscardResource(SSRTReducedSceneColor)



The UE5 rendering process captured by RenderDoc, where the red boxes are the steps related to UE5.

6.4.3.4 Nanite Clipping

Nanite's instantiation culling is handled by FInstanceCullingManager, which runs through

FDeferredShadingSceneRenderer::Renderthe entire process. The following are the definitions and declarations of it and related types:

```
// Engine\Source\Runtime\Engine\Public\SceneManagement.h

// Instantiated clipping manageme;nt resources are used toFInstanceCullingManagermiddle.
classFInstanceCullingManagerResources: public FRenderResource {

public:
    //The maximum number of indirect drawing instances is1024*1024=104Ten thousand.
    staticconstexpr uint32 MaxIndirectInstances =1024*1024;


    //Initialization and releaseRHIresource.
    virtualvoid InitRHI() override; virtualvoid
    ReleaseRHI() override;
```

```
        //Get the data interface.
        FRHIBuffer*GetInstancesIdBuffer()const{returnInstanceIdsBuffer.Buffer; } FRHIShaderResourceView*
        GetInstancesIdBufferSrv()const{return InstanceIdsBuffer.SRV.GetReference(); }


        FRHIShaderResourceView*GetPageInfoBufferSrv()const{return
        PageInfoBuffer.SRV.GetReference(); }
        FUnorderedAccessViewRHIRefGetInstancesIdBufferUav()const{return
        InstanceIdsBuffer.UAV; }
        FUnorderedAccessViewRHIRefGetPageInfoBufferUav()const{returnPageInfoBuffer.UAV; }

private:
        FRWBuffer PageInfoBuffer;//Page information buffer.
        FRWBuffer InstanceIdsBuffer;//InstantiationIDbuffer.
};

//GlobalFInstanceCullingManagerResourcesObject.
externENGINE_API TGlobalResource<FInstanceCullingManagerResources>
GInstanceCullingManagerResources;


// Engine\Source\Runtime\Renderer\Private\InstanceCulling\InstanceCullingManager.h

//Instantiate and clip intermediate data.
classFInstanceCullingIntermediate {

public:
        //Each registered view corresponds to eachInstanceVisibility bit,
                                                                    It isCullInstancesInterface processing.
        FRDGBufferRef VisibleInstanceFlags = nullptr; //All instancesI
        Expand          DThe write offset used by the extension is used in the global instance buffer   Idis  Allocate space in      quilt CullInstancesInitialize to0.
        FRDGBufferRef InstanceIdOutOffsetBuffer = nullptr;

        //The number of instances.
        int32 NumInstances =0;
        //Number of views.
        int32 NumViews =0;
};

//Instantiate the clipping result.
structFInstanceCullingResult {

        //Indirect drawing parameter buffer.
        FRDGBufferRef DrawIndirectArgsBuffer = nullptr;
        //InstantiationIDOffset buffer.
        FRDGBufferRef InstanceIdOffsetBuffer = nullptr;

        //    Get drawing parameters toFInstanceCullingDrawParamsmiddle.
        void GetDrawParameters(FInstanceCullingDrawParams &OutParams)const {

                OutParams.DrawIndirectArgsBuffer        =   DrawIndirectArgsBuffer;
                OutParams.InstanceIdOffsetBuffer        =   InstanceIdOffsetBuffer;
        }

        //Get drawing parameters with detection.
        static void CondGetDrawParameters(constFInstanceCullingResult* InstanceCullingResult,
FInstanceCullingDrawParams& OutParams)
        {
                if(InstanceCullingResult) {
```

```
            InstanceCullingResult->GetDrawParameters(OutParams);
        }
        else
        {
            OutParams.DrawIndirectArgsBuffer    =   nullptr;
            OutParams.InstanceIdOffsetBuffer    =   nullptr;
        }
    }
}   ;

//Manages the allocation of indirect parameters and clipping jobs for all instance drawing, usingGPUSceneCrop.
classFInstanceCullingManager {

public:
    FInstanceCullingManager(FInstanceCullingManagerResources& InResources,bool bInIsEnabled);


    //The maximum average number of instances after the primitive is expanded.
    staticconstexpr uint32 MaxAverageInstanceFactor =128;


    bool IsEnabled()const{returnbIsEnabled; }
    //Register the view that needs to be clipped and return the view'sid.
    int32RegisterView(constNanite::FPackedViewParams& Params); int32
    RegisterView(constFViewInfo& ViewInfo);


    //The cropping instance needs to be initialized and registered after the view is    GPUSceneAfter being updated and before rendering commands are submitted.
    void CullInstances(FRDGBuilder& GraphBuilder, FGPUScene& GPUScene);
    //Depend onCullInstancesPadding, which is used when performing final clipping
    and rendering. FInstanceCullingIntermediate CullingIntermediate;


private:
    FInstanceCullingManagerResources& Resources;
    TArray<Nanite::FPackedView> CullingViews; bool
    bIsEnabled;

    (....)
};
```

Next, analyzeFInstanceCullingManager::CullInstances the code:

```
// Engine\Source\Runtime\Renderer\Private\InstanceCulling\InstanceCullingManager.cpp

void FInstanceCullingManager::CullInstances(FRDGBuilder& GraphBuilder, FGPUScene& GPUScene)

{
#ifGPUCULL_TODO
    //Get the view and the number of instantiations.
    int32 NumViews = CullingViews.Num();
    int32 NumInstances = GPUScene.InstanceDataAllocator.GetMaxSize(); RDG_EVENT_SCOPE(GraphBuilder,
    "CullInstances [%d Views X %d Instances]", NumViews,

NumInstances);

    (.....)

    TArray<uint32> NullArray;
    NullArray.AddZeroed(1);
```

```
	//Initialize clipping intermediate dataCullingIntermediate. CullingIntermediate.InstanceIdOutOffsetBuffer =
	CreateStructuredBuffer(GraphBuilder, TEXT("InstanceCulling.OutputOffsetBufferOut"), NullArray);


	int32 NumInstanceFlagWords = FMath::DivideAndRoundUp(NumInstances, int32(sizeof
(uint32) *8));

	CullingIntermediate.NumInstances = NumInstances;
	CullingIntermediate.NumViews = NumViews;

	if(NumInstances && NumViews)//The number of views and the number of instantiations are both greater than0Only need GPUCrop.
	{
		//Create a buffer for each instance of each view to record a bit,
		CullingIntermediate.VisibleInstanceFlags =
GraphBuilder.CreateBuffer(FRDGBufferDesc::CreateStructuredDesc(sizeof(uint32), NumInstanceFlagWords *
NumViews), TEXT("InstanceCulling.VisibleInstanceFlags"));
		FRDGBufferUAVRef VisibleInstanceFlagsUAV =
		GraphBuilder.CreateUAV(CullingIntermediate.VisibleInstanceFlags);

		if(CVarCullInstances.GetValueOnRenderThread() !=0) {

			//CleaningUAV.
			AddClearUAVPass(GraphBuilder, VisibleInstanceFlagsUAV,                0);

			//Handling clipping instancesCofSParameters .
			FCullInstancesCs::FParameters* PassParameters =
GraphBuilder.AllocParameters<FCullInstancesCs::FParameters>();

			//fromGPUSceneGet instantiation and graph metadata.
			PassParameters->GPUSceneInstanceSceneData                PassPara=meGtPerUsS-
			>ceGnPeU.ISncsetnaenPcreiDmaitiavBeuSfcfenr.eSDRaVt;a                PassParameters-
			>InstanceDataSOAStride      =      GPUScene.Inst=aGncPeUDSacteanSeO.PArSimtriidtiev;e
			PBausfsfePra.rSaRmV;eters-
			PassParameters->NumInstanceFlagWords = NumInstanceFlagWords;

			//GPUsideViewThe type isNanite::FPackedView.
			//GPUsideInViewsThe type isStructuredBuffer< Nanite::FPackedView >.
			PassParameters->InViews =
GraphBuilder.CreateSRV(CreateStructuredBuffer(GraphBuilder, TEXT(
"InstanceCulling.CullingViews"),                CullingViews));
			PassParameters->NumViews =      NumViews;

			//Buffer to store visibility results.
			PassParameters->InstanceVisibilityFlagsOut = VisibleInstanceFlagsUAV;

			//CSThe one used isFCullInstancesCs,We will analyze it later.
			autoComputeShader = GetGlobalShaderMap(GMaxRHIFeatureLevel)-
> GetShader<FCullInstancesCs>();

			//Increase the clippingCS Pass.
			FComputeShaderUtils::AddPass(
				GraphBuilder,     RDG_EVENT_NAME("CullInstancesCs"),
				ComputeShader,                PassParameters,
				FComputeShaderUtils::GetGroupCount(NumInstances,


FCullInstancesCs::NumThreadsPerGroup)
```

```
                ) ;
            }
            else//The number of views and the number of instantiations are0 {


                //Everything is cleaned up to be visible.
                AddClearUAVPass(GraphBuilder,          VisibleInstanceFlagsUAV,          0xFFFFFFFF);

            }
        }
#endif//GPUCULL_TODO }
```

The above logic is to construct the parameters of the clipping shader FCullInstancesCs and call FComputeShaderUtils::AddPass to perform clipping. Let's continue to analyze the code of FCullInstancesCs:

```
// Engine\Source\Runtime\Renderer\Private\InstanceCulling\InstanceCullingManager.cpp

classFCullInstancesCs: public FGlobalShader {

    DECLARE_GLOBAL_SHADER(FCullInstancesCs);
    SHADER_USE_PARAMETER_STRUCT(FCullInstancesCs, public:    FGlobalShader)

    staticconstexpr int32 NumThreadsPerGroup =64;
    static bool ShouldCompilePermutation(constFGlobalShaderPermutationParameters& Parameters)

    {
        returnUseGPUScene(Parameters.Platform);
    }
    static                  void            M odify Compilation Environment(c
Parameters, FShaderCompilerEnvironment& OutEnvironment)
    {
        FGlobalShader::ModifyCompilationEnvironment(Parameters, OutEnvironment);
        OutEnvironment.SetDefine(TEXT("INDIRECT_ARGS_NUM_WORDS"),
        FInstanceCullingContext::IndirectArgsNumWords);
        OutEnvironment.SetDefine(TEXT("VF_SUPPORTS_PRIMITIVE_SCENE_DATA"),                1);
        OutEnvironment.SetDefine(TEXT("USE_GLOBAL_GPU_SCENE_DATA"),          1);
        OutEnvironment.SetDefine(TEXT("NUM_THREADS_PER_GROUP"),          NumThreadsPerGroup);
        OutEnvironment.SetDefine(TEXT("NANITE_MULTI_VIEW"),1);

    }

    //Declare the parameters that the shader needs to use.

    BEGIN_SHADER_PARAMETER_STRUCT(FParameters, )
        SHADER_PARAMETER_SRV(StructuredBuffer<float4>,          GPUSceneInstanceSceneData)
        SHADER_PARAMETER_SRV(StructuredBuffer<float4>,          GPUScenePrimitiveSceneData)
        SHADER_PARAMETER(uint32, InstanceDataSOAStride)

        SHADER_PARAMETER_RDG_BUFFER_SRV(StructuredBuffer< Nanite::FPackedView >, InViews)

        //Buffer to store visibility results.
        SHADER_PARAMETER_RDG_BUFFER_UAV(RWStructuredBuffer<uint>,
        InstanceVisibilityFlagsOut)

        SHADER_PARAMETER(int32,     NumInstances)
        SHADER_PARAMETER(int32,     NumInstanceFlagWords)
        SHADER_PARAMETER(int32,     NumViews)
        END_SHADER_PARAMETER_STRUCT()
```

```
};

//Implement a shader.
IMPLEMENT_GLOBAL_SHADER(FCullInstancesCs, "/Engine/Private/
InstanceCulling/CullInstances.usf",
                                                    "CullInstancesCs",          SF_Compute);
```

From the last sentence of the macro implementation above, we can see that the shader code file called by FCullInstancesCs is CullInstances.usf. Analysis:

```
// Engine\Shaders\Private\InstanceCulling\CullInstances.usf

#include    "../Common.ush"
#include    "../SceneData.ush"
#include    "../Nanite/NaniteDataDecode.ush" "../
#include    Nanite/HZBCull.ush"

RWStructuredBuffer<uint> uint    InstanceVisibilityFlagsOut;
        NumInstances;
uint NumInstanceFlagWords;
uint NumViews;
uint InstanceDataSOAStride;

//Crop the instance primary entrance.
[numthreads(NUM_THREADS_PER_GROUP,1,1)]
void CullInstancesCs(uint InstanceId : SV_DispatchThreadID) {

    //PreventionInstanceIdCrossing the line.
    if(InstanceId >= NumInstances) {


        return;
    }

    const boolbNearClip =true;

    // UnzipInstanceData MaskandOffset.
    FInstanceSceneData InstanceData = GetInstanceData(InstanceId, InstanceDataSOAStride); uint WordMask =1U<<
    (InstanceId %32U); uint InstanceWordOffset = InstanceId /32U;



    //Determine if it is effective:PrimitiveIdis not the maximum value and the length of the local bounding box is not0.
    boolbIsValid = InstanceData.PrimitiveId !=0xFFFFFFFFu && dot(InstanceData.LocalBoundsExtent,
InstanceData.LocalBoundsExtent) >0.0f;

    //Traverse allview,EachviewThe view frustum is tested for intersection with the instance's bounding box.
    for(uint ViewId =0; ViewId < NumViews; ++ViewId) {

        uint Flag = WordMask; if
        (bIsValid)
        {
            FNaniteView NaniteView = GetNaniteView(ViewId);

            //Compute the local to clip space transformation matrix.

            float4x4 LocalToTranslatedWorld = InstanceData.LocalToWorld; LocalToTranslatedWorld[3
            ].xyz += NaniteView.PreViewTranslation.xyz; float4x4 LocalToClip =
            mul(LocalToTranslatedWorld,
NaniteView.TranslatedWorldToClip);
```

```
                //Cube and frustum intersection detection.
                FFrustumCullData Cull = BoxCullFrustum(InstanceData.LocalBoundsCenter,
InstanceData.LocalBoundsExtent, LocalToClip, bNearClip,false);

                if(!Cull.bIsVisible)
                {
                    Flag =0U;
                }
            }

            //If the instance is visible, set InstanceVisibilityFlagsOutThe value at the corresponding position is1.
            if(Flag !=0U) {

                uint WordOffset = NumInstanceFlagWords * ViewId + InstanceWordOffset; //NoticeCSAtomic
                operations need to be calledInterlockXXXInterfaces, avoiding race conditions.
                InterlockedOr(InstanceVisibilityFlagsOut[WordOffset], Flag);
            }
        }
}
```

With the VisibleInstanceFlags visibility data, subsequent Pass drawing can dynamically generate drawing instructions and drawing parameters based on it to achieve GPU clipping and driving rendering pipeline.

6.4.3.5 Nanite Rasterization

Nanite rasterization mainly builds and initializes an instance of FCullingContext for each View, then calls CullRasterize, stores the rasterization results, and builds HZB. The key code is as follows:

```
for(int32 ViewIndex =0; ViewIndex < Views.Num(); ViewIndex++) {

    constFViewInfo& View = Views[ViewIndex];

    //Initialize the clipping context.
    Nanite::FCullingContext CullingContext = Nanite::InitCullingContext(
        GraphBuilder, *Scene,
        !bIsEarlyDepthComplete ? View.PrevViewInfo.NaniteHZB : View.PrevViewInfo.HZB, View.ViewRect,

        bTwoPassOcclusion, bUpdateStreaming, bSupportsMultiplePasses, bForceHWRaster, bPrimaryContext,
        bDiscardNonMoving);

    staticFString EmptyFilterName = TEXT("");
    const boolbExtractStats = Nanite::IsStatFilterActive(EmptyFilterName);

    Nanite::FPackedView PackedView = Nanite::CreatePackedViewFromViewInfo(View, RasterTextureSize,
VIEW_FLAG_HZBTEST,3);

    //Rasterization with clipping.
    Nanite::CullRasterize(GraphBuilder, *Scene, { PackedView }, CullingContext, RasterContext,
RasterState, nullptr, bExtractStats);

    Nanite::FRasterResults& RasterResults = NaniteRasterResults[ViewIndex];

    //  Rendering Rendering in advance.
    if  (bNeedsPrePass)
```

```
        {
            Nanite::EmitDepthTargets(GraphBuilder, *Scene, Views[ViewIndex], CullingContext.SOAStrides,
CullingContext.VisibleClustersSWHW, CullingContext.ViewsBuffer, SceneTextures.Depth.Target,


RasterContext.VisBuffer64,RasterResults.MaterialDepth,RasterResults.NaniteMask,RasterResults.VelocityBuffer,bNeedsPrePass);
        }

        //BuildHZB.
        if(!bIsEarlyDepthComplete && bTwoPassOcclusion && View.ViewState) {

            RDG_EVENT_SCOPE(GraphBuilder,"Nanite::BuildHZB");

            FRDGTextureRef SceneDepth = SystemTextures.Black;
            FRDGTextureRef GraphHZB = nullptr;

            BuildHZBFurthest(GraphBuilder,SceneDepth, RasterContext.VisBuffer64, PrimaryViewRect,
FeatureLevel, ShaderPlatform, TEXT("Nanite.HZB"), &GraphHZB );

            GraphBuilder.QueueTextureExtraction( GraphHZB, &View.ViewState-
> PrevFrameViewInfo.NaniteHZB ); }


        //Extract rasterization and clipping results.
        Nanite::ExtractResults(GraphBuilder, CullingContext, RasterContext, RasterResults);
}
```

Focus on analyzing the code of Nanite::CullRasterize:

```
// Engine\Source\Runtime\Renderer\Private\Nanite\NaniteRender.cpp

void CullRasterize(
        FRDGBuilder& GraphBuilder, const
        FScene& Scene,
        constTArray<FPackedView, SceneRenderingAllocator>& Views, uint32
        NumPrimaryViews,                    // Number of non-mip views
        FCullingContext&                    CullingContext,
        const    FRasterContext& RasterContext,
        const    FRasterState& RasterState,
        constTArray<FInstanceDraw, SceneRenderingAllocator>* OptionalInstanceDraws,
        // VirtualShadowMapArray is the supplier of virtual to physical translation, probably could abstract this a bit better,

        FVirtualShadowMapArray* VirtualShadowMapArray, bool
        bExtractStats
)
{
        //If there are too many views, split them into multiplePassDe-rasterize. Onlydepth-only
        Rendering can happen. if(Views.Num() > MAX_VIEWS_PER_CULL_RASTERIZE_PASS) {

            CullRasterizeMultiPass(GraphBuilder, Scene, Views, NumPrimaryViews, CullingContext, RasterContext,
RasterState, OptionalInstanceDraws, VirtualShadowMapArray, bExtractStats);

            return;
        }

        RDG_EVENT_SCOPE(GraphBuilder,"Nanite::CullRasterize");
```

```
(……)

//Creates a structured buffer for a view.
{
    constuint32 ViewsBufferElements = FMath::RoundUpToPowerOfTwo(Views.Num());
    CullingContext.ViewsBuffer = CreateStructuredBuffer(GraphBuilder,
TEXT("Nanite.Views"), Views.GetTypeSize(), ViewsBufferElements, Views.GetData(), Views.Num() *
Views.GetTypeSize());
}

//  Structured buffers for handling clipping context.
if  (OptionalInstanceDraws)
{
    constuint32 InstanceDrawsBufferElements =
FMath::RoundUpToPowerOfTwo(OptionalInstanceDraws->Num());
    CullingContext.InstanceDrawsBuffer = CreateStructuredBuffer (

        GraphBuilder,
        TEXT("Nanite.InstanceDraws"),
        OptionalInstanceDraws->GetTypeSize(),
        InstanceDrawsBufferElements,
        OptionalInstanceDraws->GetData(),
        OptionalInstanceDraws->Num() *
                                        OptionalInstanceDraws->GetTypeSize()
    ) ;
    CullingContext.NumInstancesPreCull      =  OptionalInstanceDraws->Num();
}
else
{
    CullingContext.InstanceDrawsBuffer      =nullptr;
    CullingContext.NumInstancesPreCull      =
    Scene.GPUScene.InstanceDataAllocator.GetMaxSize();
}

(……)

//Cropping parameters.
FCullingParameters      CullingParameters;
{
    CullingParameters.InViews                   =
    GraphBuilder.CreateSRV(CullingContext.ViewsBuffer);
    CullingParameters.NumViews CullingParamete=rsV.NieuwmsP.Nriumma(r)y;Views
    CullingParameters.DisocclusionLodScaleFactor= = NGuNmanPirtiemDaisroyVccieluwssio;nHack ?0.01f:

1.0f;       //TODO:Get rid of this hack
    CullingParameters.HZBTexture            =
RegisterExternalTextureWithFallback(GraphBuilder,               CullingContext.PrevHZB,
GSystemTextures.BlackDummy);
    CullingParameters.HZBSize               =CullingContext.PrevHZB ?
CullingContext.PrevHZB->GetDesc().Extent    : FVector2D(0.0f);
    CullingParameters.HZBSampler            = TStaticSamplerState< SF_Point, AM_Clamp,
AM_Clamp, AM_Clamp >::GetRHI();
    CullingParameters.SOAStrides            = CullingContext.SOAStrides;
    CullingParameters.MaxCandidateClusters                  =
    Nanite::FGlobalResources::GetMaxCandidateClusters();
    CullingParameters.MaxVisibleClusters                    =
  Nanite::FGlobalResources::GetMaxVisibleClusters();
     CullingParameters.RenderFlags          =CullingContext.RenderFlags;
     CullingParameters.DebugFlags           = CullingContext.DebugFlags;
```

```
            CullingParameters.CompactedViewInfo = nullptr;
            CullingParameters.CompactedViewsAllocation = nullptr;
        }

        FVirtualTargetParameters VirtualTargetParameters;
        //deal withVSM(Virtual shadow map)
        array. if(VirtualShadowMapArray) {

            VirtualTargetParameters.VirtualShadowMap                =VirtualShadowMapArray-
> GetUniformBuffer(GraphBuilder);
VirtualTargetParameters.PageFlags                        = GraphBuilder.CreateSRV(VirtualShadowMapArray-
> PageFlagsRDG, PF_R32_UINT);
VirtualTargetParameters.HPageFlags                        = GraphBuilder.CreateSRV(VirtualShadowMapArray-
> HPageFlagsRDG, PF_R32_UINT); VirtualTargetParameters.PageRectBounds
            GraphBuilder.CreateSRV(VirtualShadowMapArray->Pa=geRectBoundsRDG);


            //    If provided from the previous frameHZB,    The previous frame is also needed    Pagesurface.
            FRDGBufferRef HZBPageTableRDG = VirtualShadowMapArray->PageTableRDG; if
            (CullingContext.PrevHZB) {

                check(VirtualShadowMapArray->CacheManager); TRefCountPtr<FRDGPooledBuffer>
                HZBPageTable = VirtualShadowMapArray-
> CacheManager->PrevBuffers.PageTable;
                check( HZBPageTable );
                HZBPageTableRDG = GraphBuilder.RegisterExternalBuffer( HZBPageTable, TEXT(
"Shadow.Virtual.HZBPageTable") );
            }
            V irtual T arget Parameters . S hadow HZBP age T HZBPageTableRDG, PF_R32_UINT );                                    ab

        }

        //deal withGPUScenedata. FGPUSceneParameters GPUSceneParameters;
        GPUSceneParameters.GPUSceneInstanceSceneData GPUSceneParameters.GPUScenePrimitiveSceneData
        GPUSceneParameters.GPUSceneFrameNumber = Scene.GPUS=ceSnce.nGee.tGSPceUnSecFernaem.IneNstuamncbeeDra()t;
                                                                = Scene.GPUScene.PrimitiveBuffer.SRV;


        //CroppingVSM.
        if(VirtualShadowMapArray && CVarCompactVSMViews.GetValueOnRenderThread() !=0) {

            RDG_GPU_STAT_SCOPE(GraphBuilder, NaniteInstanceCullVSM);

            //Compact views to remove unnecessary (empty)mipView, need to beGPUDo it becauseGPUI just found outmipWhich ones do you have?    page.
            constuint32 ViewsBufferElements = FMath::RoundUpToPowerOfTwo(Views.Num()); FRDGBufferRef
            CompactedViews =
GraphBuilder.CreateBuffer(FRDGBufferDesc::CreateStructuredDesc(sizeof(FPackedView), ViewsBufferElements),
                                TEXT("Shadow.Virtual.CompactedViews"));
            FRDGBufferRef      CompactedViewInfo =
GraphBuilder.CreateBuffer(FRDGBufferDesc::CreateStructuredDesc(sizeof(FCompactedViewInfo), Views.Num()), TEXT(
"Shadow.Virtual.CompactedViewInfo"));

            const staticuint32 TheZeros[2] = {0U,0U};
            FRDGBufferRef CompactedViewsAllocation = CreateStructuredBuffer(GraphBuilder, TEXT(
"Shadow.Virtual.CompactedViewsAllocation"),sizeof(uint32),2, TheZeros, sizeof(TheZeros),
ERDGInitialDataFlags::NoCopy);
            {
                FCompactViewsVSM_CS::FParameters* PassParameters =
```

```
GraphBuilder.AllocParameters< FCompactViewsVSM_CS::FParameters >();

                PassParameters->GPUSceneParameters          =  GPUSceneParameters;
                PassParameters->CullingParameters           =  CullingParameters;
                PassParameters->VirtualShadowMap            =  VirtualTargetParameters;

                PassParameters->CompactedViewsOut =          GraphBuilder.CreateUAV(CompactedViews); =
                PassParameters->CompactedViewInfoOut
GraphBuilder.CreateUAV(CompactedViewInfo);
                PassParameters->CompactedViewsAllocationOut              =
GraphBuilder.CreateUAV(CompactedViewsAllocation);

                autoComputeShader = CullingContext.ShaderMap->GetShader<FCompactViewsVSM_CS>
();

                //UtilizationCPressSShrink and cropVSM.
                FComputeShaderUtils::AddPass(
                    GraphBuilder,      RDG_EVENT_NAME("CompactViewsVSM"),
                    ComputeShader,                                PassParameters,
                    FComputeShaderUtils::GetGroupCount(NumPrimaryViews,

                                                                                    64)
                ) ;
            }

            //Overwrite the original information with the compressed view.
            CullingParameters.InViews CullingC=onGterxatp.VhiBeuwilsdBeurf.Cferre ateSRV(CompactedViews);
            CullingParameters.CompactedViewIn=fCoo mpactedViews;
            CullingParameters.CompactedViewsAllocation    = GraphBuilder.CreateSRV(CompactedViewInfo);
            GraphBuilder.CreateSRV(CompactedViewsAllocation);       =

        }

        //Initialize the parameters of the clipping context.
        {
            FInitArgs_CS::FParameters* PassParameters = GraphBuilder.AllocParameters<
FInitArgs_CS::FParameters >();

            PassParameters->RenderFlags = CullingParameters.RenderFlags;

            PassParameters->OutMainAndPostPassPersistentStates                  =  GraphBuilder.CreateUAV(
            CullingContext.MainAndPostPassPersistentStates );
            PassParameters->InOutMainPassRasterizeArgsSWHW                      =  GraphBuilder.CreateUAV(
            CullingContext.MainRasterizeArgsSWHW );

            uint32 ClampedDrawPassIndex = FMath::Min(CullingContext.DrawPassIndex,2u);

            if(CullingContext.bTwoPassOcclusion) {

                PassParameters->OutOccludedInstancesArgs              =  GraphBuilder.CreateUAV(
CullingContext.OccludedInstancesArgs );
                PassParameters->InOutPostPassRasterizeArgsSWHW              = GraphBuilder.CreateUAV(
CullingContext.PostRasterizeArgsSWHW );
            }

            if(CullingContext.RenderFlags & RENDER_FLAG_HAVE_PREV_DRAW_DATA) {

                PassParameters->InOutTotalPrevDrawClusters =
```

```
GraphBuilder.CreateUAV(CullingContext.TotalPrevDrawClustersBuffer);
            }
            else
            {
                // Use any UAV just to keep render graph happy that something is bound, but
the shader doesn't actually touch this.
                PassParameters->InOutTotalPrevDrawClusters = PassParameters-
> OutMainAndPostPassPersistentStates; }


            FInitArgs_CS::FPermutationDomain PermutationVector;
            PermutationVector.Set<FInitArgs_CS::FOcclusionCullingDim>( CullingContext.bTwoPassOcclusion );

            PermutationVector.Set<FInitArgs_CS::FDrawPassIndexDim>( ClampedDrawPassIndex                    ) ;

            autoComputeShader = CullingContext.ShaderMap->GetShader<FInitArgs_CS>(PermutationVector);


            //Also usedCearly Initialization parameters.
            FComputeShaderUtils::AddPass(
                GraphBuilder,
                RDG_EVENT_NAME("InitArgs"          ) ,
                ComputeShader,
                PassParameters,
                FIntVector(1,1,1)
            ) ;
        }

//          Allocate candidate buffers, the life cycle is onlyCullRasterizeperiod.
FRDGBufferRef MainCandidateNodesAndClustersBuffer = nullptr; FRDGBufferRef
PostCandidateNodesAndClustersBuffer = nullptr;
AllocateCandidateBuffers(GraphBuilder,                CullingContext.ShaderMap,
&MainCandidateNodesAndClustersBuffer,              CullingContext.bTwoPassOcclusion nullptr);  ?
&PostCandidateNodesAndClustersBuffer                :

        //Instantiation level andClusterCropping, including no occlusionPassOr block the
        mainPass. AddPass_InstanceHierarchyAndClusterCull(
            GraphBuilder, Scene, CullingParameters, Views, NumPrimaryViews, CullingContext,
            RasterContext, RasterState, GPUSceneParameters,
            MainCandidateNodesAndClustersBuffer,
            PostCandidateNodesAndClustersBuffer, CullingContext.bTwoPassOcclusion ?
            CULLING_PASS_OCCLUSION_MAIN :



CULLING_PASS_NO_OCCLUSION,
            VirtualShadowMapArray,
            VirtualTargetParameters
        ) ;

        //Rasterization.
        AddPass_Rasterize(
            GraphBuilder,
            Views,
```

```
            RasterContext,                                  RasterState,
            CullingContext.SOAStrides,
            CullingContext.RenderFlags,
            CullingContext.ViewsBuffer,
            CullingContext.VisibleClustersSWHW, nullptr,

            CullingContext.SafeMainRasterizeArgsSWHW,
            CullingContext.TotalPrevDrawClustersBuffer,
            GPUSceneParameters,                                 true,
            VirtualShadowMapArray,
            VirtualTargetParameters


    ) ;

    //Occlusion postPass.Re-detect instances that were not visible in the previous frame andCluster, if(CullingContext.bTwoPassOcclusion) {   If they are visible in this frame, render them .

            //Use the occluder from the previous frame to create a nearestHkDJ, to detect the remaining occluders again. {

                RDG_EVENT_SCOPE(GraphBuilder,"BuildPreviousOccluderHZB");

                FSceneTextureParameters SceneTextures =
GetSceneTextureParameters(GraphBuilder);

                FRDGTextureRef SceneDepth = SceneTextures.SceneDepthTexture;
                FRDGTextureRef RasterizedDepth = RasterContext.VisBuffer64;

                if( RasterContext.RasterTechnique == ERasterTechnique::DepthOnly ) {

                    SceneDepth     = GraphBuilder.RegisterExternalTexture();
GSystemTextures.BlackDummy
                    RasterizedDepth        = RasterContext.DepthBuffer;
                }

                FRDGTextureRef OutFurthestHZBTexture;

                FIntRectViewRect(0,0, RasterContext.TextureSize.X,
RasterContext.TextureSize.Y);
                if(Views.Num() ==1) {

                    ViewRect = FIntRect(Views[0].ViewRect.X, Views[0].ViewRect.Y,
Views[0].ViewRect.Z, Views[0].ViewRect.W);
                }

                //BuildHZB.
                BuildHZBFurthest(
                GraphBuilder,
                SceneDepth,
                RasterizedDepth,
                CullingContext.HZBBuildViewRect,
                Scene.GetFeatureLevel(),
                Scene.GetShaderPlatform(), TEXT(
                "Nanite.PreviousOccluderHZB"),
                /* OutFurthestHZBTexture = */&OutFurthestHZBTexture);


                CullingParameters.HZBTexture            = OutFurthestHZBTexture;
                CullingParameters.HZBSize =          CullingParameters.HZBTexture->Desc.Extent;
```

```
            }

            //PostPass.
            AddPass_InstanceHierarchyAndClusterCull(
                GraphBuilder,                                   Scene,
                CullingParameters,                              Views,
                NumPrimaryViews,                CullingContext,
                RasterContext,                      RasterState,
                GPUSceneParameters,
                MainCandidateNodesAndClustersBuffer,
                PostCandidateNodesAndClustersBuffer,
                CULLING_PASS_OCCLUSION_POST,
                VirtualShadowMapArray,
                VirtualTargetParameters




            ) ;

            //Rendering PostPass.
            AddPass_Rasterize(
                GraphBuilder,           Views,          RasterContext,
                RasterState,            CullingContext.SOAStrides,
                CullingContext.RenderFlags,     CullingContext.ViewsBuffer,
                CullingContext.VisibleClustersSWHW,
                CullingContext.MainRasterizeArgsSWHW,
                CullingContext.SafePostRasterizeArgsSWHW,
                CullingContext.TotalPrevDrawClustersBuffer,
                GPUSceneParameters,
                                                                false,
                VirtualShadowMapArray,
                VirtualTargetParameters




            ) ;
        }

        if(RasterContext.RasterTechnique != ERasterTechnique::DepthOnly) {

            //PreviousPassRenderedClusterThe index and count have nothing to do with depth-only
            rendering. CullingContext.DrawPassIndex++; CullingContext.RenderFlags |=
            RENDER_FLAG_HAVE_PREV_DRAW_DATA;

        }

        (.....)
}
```

Next, we will focus on the two interfaces AddPass_InstanceHierarchyAndClusterCull and AddPass_Rasterize. First, AddPass_InstanceHierarchyAndClusterCull:

```
void AddPass_InstanceHierarchyAndClusterCull(
        FRDGBuilder& GraphBuilder, const
        FScene& Scene,
```

```
        const FCullingParameters& CullingParameters, const
        TArray<FPackedView, SceneRenderingAllocator>& Views,const
        uint32NumPrimaryViews,constFCullingContext&
        CullingContext,constFRasterContext& RasterContext,const FRasterState&
        RasterState,constFGPUSceneParameters &GPUSceneParameters,
                                                                FRDGBufferRef
        MainCandidateNodesAndClusters,                          FRDGBufferRef
        PostCandidateNodesAndClusters,          uint32          Culling Pass,
        FVirtualShadowMapArray                  * VirtualShadowMapArray,
        FVirtualTargetParameters        &VirtualTargetParameters        )



{
        (......)

        const boolbMultiView = Views.Num() >1|| VirtualShadowMapArray != nullptr;

        if(VirtualShadowMapArray) {

                (......)
        }
        //Handles instanced culling.
        else if(CullingContext.NumInstancesPreCull >0|| CullingPass ==
        CULLING_PASS_OCCLUSION_POST)
        {
                RDG_GPU_STAT_SCOPE(GraphBuilder, NaniteInstanceCull);

                //Handling instanced cullingCSParameters.
                FInstanceCull_CS::FParameters* PassParameters = GraphBuilder.AllocParameters<
FInstanceCull_CS::FParameters >();

                PassParameters->NumInstances                                    =
                CullingContext.NumInstancesPreCull;
                PassParameters->MaxNodes                                        =
                Nanite::FGlobalResources::GetMaxNodes();
                PassParameters->ImposterMaxPixels                                 =  GNaniteImposterMaxPixels;

                PassParameters->GPUSceneParameters          = GPUSceneParameters;
                PassParameters->RasterParameters          =  RasterContext.Parameters;
                PassParameters->CullingParameters           =CullingParameters;

                constERasterTechnique Technique = RasterContext.RasterTechnique; PassParameters-
                >OnlyCastShadowsPrimitives = Technique == ERasterTechnique::DepthOnly ?1:0;


                PassParameters->ImposterAtlas = Nanite::GStreamingManager.GetRootPagesSRV();

                PassParameters->OutMainAndPostPassPersistentStates                  = GraphBuilder.CreateUAV(
                CullingContext.MainAndPostPassPersistentStates );

                if(CullingContext.StatsBuffer) {

                        PassParameters->OutStatsBuffer                                  =
GraphBuilder.CreateUAV(CullingContext.StatsBuffer);
                }

                //Set different parameters according to different cropping methods.
```

```
if(CullingPass == CULLING_PASS_NO_OCCLUSION ) {

    if(CullingContext.InstanceDrawsBuffer) {

        PassParameters->InInstanceDraws = GraphBuilder.CreateSRV(
CullingContext.InstanceDrawsBuffer );
    }
    P ass Parameters -> O ut C a =nGrdapihBd uiladetr.eCreNatoeUdAV(es A nd C l
MainCandidateNodesAndClusters);
}
else if( Culling Pass = = CULLING _ PASS _ O C C
{
    PassParameters->OutOccludedInstances = GraphBuilder.CreateUAV(
CullingContext.OccludedInstances );
    PassParameters->OutOccludedInstancesArgs = GraphBuilder.CreateUAV(
CullingContext.OccludedInstancesArgs );
    PassParameters->OutCandidateNodesAndClusters = GraphBuilder.CreateUAV(
MainCandidateNodesAndClusters );
}
else
{
    PassParameters->InInstanceDraws = GraphBuilder.CreateSRV(
CullingContext.OccludedInstances );
    PassParameters->InOccludedInstancesArgs = GraphBuilder.CreateSRV(
CullingContext.OccludedInstancesArgs );
    PassParameters->OutCandidateNodesAndClusters = GraphBuilder.CreateUAV(
PostCandidateNodesAndClusters);
}

check(CullingContext.ViewsBuffer);

//Processing of permutation parameters.
constuint32 InstanceCullingPass = CullingContext.InstanceDrawsBuffer != nullptr ?
CULLING_PASS_EXPLICIT_LIST : CullingPass;
FInstanceCull_CS::FPermutationDomain PermutationVector;
PermutationVector.Set<FInstanceCull_CS::FCullingPassDim>(InstanceCullingPass);
PermutationVector.Set<FInstanceCull_CS::FMultiViewDim>(bMultiView);
PermutationVector.Set<FInstanceCull_CS::FNearClipDim>(RasterState.bN earClip); PermutationVector.Set
<FInstanceCull_CS::FDebugFlagsDim>(CullingContext.DebugFlags
!=0);
PermutationVector.Set<FInstanceCull_CS::FRasterTechniqueDim>
(int32(RasterContext.RasterTechnique));

autoComputeShader = CullingContext.ShaderMap->GetShader<FInstanceCull_CS>
(PermutationVector);

//PostPassInstance cropping.
if(InstanceCullingPass == CULLING_PASS_OCCLUSION_POST) {

    PassParameters->IndirectArgs = CullingContext.OccludedInstancesArgs;
    FComputeShaderUtils::AddPass(
        GraphBuilder, RDG_EVENT_NAME("Post Pass:
        InstanceCull"), ComputeShader,
        PassParameters, PassParameters-
        >IndirectArgs, 0


    ) ;
```

```
            }
            else                //          Main channel instance cropping .
            {
                FComputeShaderUtils::AddPass(
                    GraphBuilder,
                    InstanceCullingPass == CULLING_PASS_OCCLUSION_MAIN ?                    RDG_EVENT_NAME(
"Main Pass: InstanceCull") :
                    InstanceCullingPass == CULLING_PASS_NO_OCCLUSION ?                      RDG_EVENT_NAME(
"Main Pass: InstanceCull - No occlusion") :
                                                                                            RDG_EVENT_NAME(
"Main Pass: InstanceCull - Explicit list"),
                    ComputeShader,                                                  PassParameters,
                    FComputeShaderUtils::GetGroupCount(CullingContext.NumInstancesPreCull,
                                                                                                64)
                ) ;
            }
        }

        // ClusterCrop.
        {
RDG_GPU_STAT_SCOPE(GraphBuilder, NaniteClusterCull);
FPersistentClusterCull_CS::FParameters* PassParameters GraphBuilder.Allo=cParameters<
FPersistentClusterCull_CS::FParameters                                                  > ();

            // ClusterCutting is usedGPUScene,GStreamingManager        And other parameters.
            PassParameters->GPUSceneParameters              =  GPUSceneParameters;
            PassParameters->CullingParameters               =  CullingParameters;
            PassParameters->MaxNodes                        =  Nanite::FGlobalResources::GetMaxNodes();

            PassParameters->ClusterPageHeaders              =
            Nanite::GStreamingManager.GetClusterPageHeadersSRV();
            PassParameters->ClusterPageData                 =
Nanite::GStreamingManager.GetClusterPageDataSRV();
            PassParameters->HierarchyBuffer                 =
Nanite::GStreamingManager.GetHierarchySRV();

            check(CullingContext.DrawPassIndex ==0|| CullingContext.RenderFlags &
RENDER_FLAG_HAVE_PREV_DRAW_DATA);// sanity check
            //Process the previous frame of data.
            if(CullingContext.RenderFlags & RENDER_FLAG_HAVE_PREV_DRAW_DATA) {


                PassParameters->InTotalPrevDrawClusters =
GraphBuilder.CreateSRV(CullingContext.TotalPrevDrawClustersBuffer);
            }
            else
            {
                FRDGBufferRef Dummy =
GraphBuilder.RegisterExternalBuffer(Nanite::GGlobalResources.GetStructureBufferStride8(), TEXT(
"Nanite.StructuredBufferStride8"));
                PassParameters->InTotalPrevDrawClusters = GraphBuilder.CreateSRV(Dummy);
            }

            PassParameters->MainAndPostPassPersistentStates                 = GraphBuilder.CreateUAV(
            CullingContext.MainAndPostPassPersistentStates );
            //Candidate nodes andCluster.
            if(CullingPass == CULLING_PASS_NO_OCCLUSION || CullingPass ==

CULLING_PASS_OCCLUSION_MAIN )
```

```
        {
            PassParameters->InOutCandidateNodesAndClusters = GraphBuilder.CreateUAV(
MainCandidateNodesAndClusters );
            PassParameters->VisibleClustersArgsSWHW = GraphBuilder.CreateUAV(
CullingContext.MainRasterizeArgsSWHW );

            if(CullingPass == CULLING_PASS_OCCLUSION_MAIN) {

                PassParameters->OutOccludedNodesAndClusters = GraphBuilder.CreateUAV(
PostCandidateNodesAndClusters );
            }
        }
        else
        {
            PassParameters->InOutCandidateNodesAndClusters = GraphBuilder.CreateUAV(
PostCandidateNodesAndClusters );
            PassParameters->OffsetClustersArgsSWHW = GraphBuilder.CreateSRV(
CullingContext.MainRasterizeArgsSWHW );
            PassParameters->VisibleClustersArgsSWHW = GraphBuilder.CreateUAV(
CullingContext.PostRasterizeArgsSWHW );
        }

        //OutputUAV,Contains visibleClusterand streaming requests.
        PassParameters->OutVisibleClustersSWHW = GraphBuilder.CreateUAV(
        CullingContext.VisibleClustersSWHW );
        PassParameters->OutStreamingRequests = GraphBuilder.CreateUAV(
        CullingContext.StreamingRequests );

        if(VirtualShadowMapArray) {

            PassParameters->VirtualShadowMap = VirtualTargetParameters;
            PassParameters->OutDynamicCasterFlags =
GraphBuilder.CreateUAV(VirtualShadowMapArray->DynamicCasterPageFlagsRDG, PF_R32_UINT);
        }

        if(CullingContext.StatsBuffer) {

            PassParameters->OutStatsBuffer =
GraphBuilder.CreateUAV(CullingContext.StatsBuffer);
        }

        PassParameters->LargePageRectThreshold =
        CVarLargePageRectThreshold.GetValueOnRenderThread();

        check(CullingContext.ViewsBuffer);

        //arrangement.
        FPersistentClusterCull_CS::FPermutationDomain PermutationVector;
        PermutationVector.Set<FPersistentClusterCull_CS::FCullingPassDim>(CullingPass); lipDim>


(RasterState.bNearClip);
        PermutationVector.Set<FPersistentClusterCull_CS::FVirtualTextureTargetDim>
(VirtualShadowMapArray != nullptr);
        PermutationVector.Set<FPersistentClusterCull_CS::FClusterPerPageDim>
(GNaniteClusterPerPage && VirtualShadowMapArray != nullptr);
        PermutationVector.Set<FPersistentClusterCull_CS::FDebugFlagsDim>
(CullingContext.DebugFlags !=0);
```

```
        autoComputeShader = CullingContext.ShaderMap-
> GetShader<FPersistentClusterCull_CS>(PermutationVector);

        // CS PassCall.
        FComputeShaderUtils::AddPass(
            GraphBuilder,
            CullingPass == CULLING_PASS_NO_OCCLUSION              ? RDG_EVENT_NAME("Main Pass:
PersistentCull - No occlusion") :
            CullingPass == CULLING_PASS_OCCLUSION_MAIN            ? RDG_EVENT_NAME("Main Pass:
PersistentCull) :
            RDG_EVENT_NAME("Post  Pass:  PersistentCull"),
            ComputeShader,                PassParameters,
            FIntVector(GRHIPersistentThreadGroupCount,1,
                                                                  1)
        ) ;
    }

    //Calculate rasterization parameters to ensure that subsequent rasterization passes are correct and safe.

    {
        FCalculateSafeRasterizerArgs_CS::FParameters* PassParameters =
 GraphBuilder.AllocParameters< FCalculateSafeRasterizerArgs_CS::FParameters                    > ();

        const boolbPrevDrawData             = (CullingContext.RenderFlags & 0;
        RENDER_FLAG_HAVE_PREV_DRAW_!=DATA)
        const boolbPostPass                 = (CullingPass == CULLING_PASS_OCCLUSION_POST) !=0;

        if(bPrevDrawData)
        {
            PassParameters->InTotalPrevDrawClusters                =
GraphBuilder.CreateSRV(CullingContext.TotalPrevDrawClustersBuffer);
        }

        if(bPostPass)
        {
            PassParameters->OffsetClustersArgsSWHW                 =
GraphBuilder.CreateSRV(CullingContext.MainRasterizeArgsSWHW);
            PassParameters->InRasterizerArgsSWHW                =
GraphBuilder.CreateSRV(CullingContext.PostRasterizeArgsSWHW);
            PassParameters->OutSafeRasterizerArgsSWHW =
GraphBuilder.CreateUAV(CullingContext.SafePostRasterizeArgsSWHW);
        }
        else
        {
            PassParameters->InRasterizerArgsSWHW                =
GraphBuilder.CreateSRV(CullingContext.MainRasterizeArgsSWHW);
            PassParameters->OutSafeRasterizerArgsSWHW =
GraphBuilder.CreateUAV(CullingContext.SafeMainRasterizeArgsSWHW);
        }

        PassParameters->MaxVisibleClusters                         =
        Nanite::FGlobalResources::GetMaxVisibleClusters();
        PassParameters->RenderFlags                                  = CullingContext.RenderFlags;

        FCalculateSafeRasterizerArgs_CS::FPermutationDomain PermutationVector;
        PermutationVector.Set<FCalculateSafeRasterizerArgs_CS::FHasPrevDrawData> (bPrevDrawData);

        PermutationVector.Set<FCalculateSafeRasterizerArgs_CS::FIsPostPass>(bPostPass);
```

```
            autoComputeShader = CullingContext.ShaderMap->GetShader<
FCalculateSafeRasterizerArgs_CS >(PermutationVector);

            FComputeShaderUtils::AddPass(
                  GraphBuilder,
                  bPostPass ? RDG_EVENT_NAME("Post Pass: CalculateSafeRasterizerArgs") :
RDG_EVENT_NAME("Main Pass: CalculateSafeRasterizerArgs"),
                  ComputeShader,
                  PassParameters,
                  FIntVector(1,1,1)
            ) ;
      }
}
```

The above involves multiple calls to Compute Shader. Due to space limitations, we will not analyze its shader code. The following will focus on AddPass_Rasterize:

```
void AddPass_Rasterize(
      FRDGBuilder& GraphBuilder,
      constTArray<FPackedView, SceneRenderingAllocator>& Views, const
            FRasterContext& RasterContext,
      const    FRasterState& RasterState,
      FIntVector4       SOAStrides,
      uint32 RenderFlags,
      FRDGBufferRef       ViewsBuffer,FRDGBufferRef
      FRDGBufferRef       ViewRectsBuffer,FRDGBufferRef
      FRDGBufferRef       ClusterOffsetSWHW,const
      FGPUSceneParameters&indirectArgs,
      bool bMainPass,  TotalPrevDrawClustersBuffer,
      FVirtualShadowMapArray*                GPUSceneParameters,
      FVirtualTargetParameters&

                                          VirtualShadowMapArray,
                                            VirtualTargetParameters
)
{
      (......)

      //Assign rasterization parameters.
      auto* RasterPassParameters = GraphBuilder.AllocParameters<FHWRasterizePS::FParameters>
();
      auto* CommonPassParameters = &RasterPassParameters->Common;

      //set upClusterPages and headers.
      CommonPassParameters->ClusterPageData =          GStreamingManager.GetClusterPageDataSRV(); =
      CommonPassParameters->ClusterPageHeaders
      GStreamingManager.GetClusterPageHeadersSRV();

      //    View buffer data.
      if    (ViewsBuffer)
      {
            CommonPassParameters->InViews       =  GraphBuilder.CreateSRV(ViewsBuffer);
      }

      //Plotting parameters.
      CommonPassParameters->GPUSceneParameters = GPUSceneParameters;
```

```
CommonPassParameters->RasterParameters = RasterContext.Parameters; CommonPassParameters-
>VisualizeModeBitMask = RasterContext.VisualizeModeBitMask; CommonPassParameters->SOAStrides =
SOAStrides;
CommonPassParameters->MaxVisibleClusters =
Nanite::FGlobalResources::GetMaxVisibleClusters();
CommonPassParameters->RenderFlags = if        RenderFlags;
(RasterState.CullMode == CM_CCW) {

        CommonPassParameters->RenderFlags          |= RENDER_FLAG_REVERSE_CULLING;
}
C ommon P ass P parameters -> V isible C lusters SWH GraphBuilder.CreateSRV(VisibleClustersSWHW);                    W


if(VirtualShadowMapArray) {

        CommonPassParameters->VirtualShadowMap = VirtualTargetParameters;
}

if(!bMainPass)
{
        CommonPassParameters->InClusterOffsetSWHW          =
        GraphBuilder.CreateSRV(ClusterOffsetSWHW);
}
C ommon P ass Parameters -> I ndirect A rgs                                                    =      I nd      i   r

const boolbHavePrevDrawData = (RenderFlags & RENDER_FLAG_HAVE_PREV_DRAW_DATA); if
(bHavePrevDrawData)
{
        CommonPassParameters->InTotalPrevDrawClusters =
        GraphBuilder.CreateSRV(TotalPrevDrawClustersBuffer);
}

constERasterTechniqueTechnique = RasterContext.RasterTechnique; const
ERasterScheduling Scheduling = RasterContext.RasterScheduling; const boolbNearClip =
RasterState.bNearClip;
const boolbMultiView = Views.Num() >1|| VirtualShadowMapArray != nullptr;

ERDGPassFlags ComputePassFlags = ERDGPassFlags::Compute;

//If it is a combination of software and hardware, create aSkipBarrierTaggedUAV.
if(Scheduling == ERasterScheduling::HardwareAndSoftwareOverlap) {

        constautoCreateSkipBarrierUAV = [&](auto& InOutUAV) {

                if(InOutUAV)
                {
                        //BroughtERDGUnorderedAccessViewFlags::SkipBarriermark.
                        InOutUAV = GraphBuilder.CreateUAV(InOutUAV->Desc,
ERDGUnorderedAccessViewFlags::SkipBarrier);
                }
        };

        //Create BandSkipBarrierTaggedUAV,To allow softChard rlteemap Stiery taeyforkkiOverplapB.arrivalUAV(CommonPassParameters-
        >   RasterParameters.OutDepthBuffer);          CreateSkipBarrierUAV(CommonPassParameters-
        > RasterParameters.OutVisBuffer64); >          CreateSkipBarrierUAV(CommonPassParameters-
        RasterParameters.OutDbgBuffer64);  >           CreateSkipBarrierUAV(CommonPassParameters-
        RasterParameters.OutDbgBuffer32);  >           CreateSkipBarrierUAV(CommonPassParameters-
        RasterParameters.LockBuffer);
```

```
            ComputePassFlags = ERDGPassFlags::AsyncCompute;
        }

        FIntRectViewRect(Views[0].ViewRect.X, Views[0].ViewRect.Y, Views[0].ViewRect.Z, Views[0].ViewRect.W);

        if(bMultiView)
        {
            ViewRect.Min      =  FIntPoint::ZeroValue;
            ViewRect.Max      =  RasterContext.TextureSize;
        }

        //    deal withVSM.
        if    (VirtualShadowMapArray)
        {
            ViewRect.Min = FIntPoint::ZeroValue; if
            ( GNaniteClusterPerPage ) {

                ViewRect.Max = FIntPoint( FVirtualShadowMap::PageSize,
FVirtualShadowMap::PageSize ) * FVirtualShadowMap::RasterWindowPages;
            }
            else
            {
                ViewRect.Max = FIntPoint( FVirtualShadowMap::VirtualMaxResolutionXY,
FVirtualShadowMap::VirtualMaxResolutionXY );
            }
        }

        //First use the traditional hardware rendering pipeline rasterization.

        {
            const boolbUsePrimitiveShader = UsePrimitiveShader();

            const boolbUseAutoCullingShader =
                GRHISupportsPrimitiveShaders !          &&
                bUsePrimitiveShader &&
                GNaniteAutoShaderCulling !=          0;

            //deal withVSparameter.

            FHWRasterizeVS::FPermutationDomain PermutationVectorVS;
            PermutationVectorVS.Set<FHWRasterizeVS::FRasterTechniqueDim>(int32(Technique));
            PermutationVectorVS.Set<FHWRasterizeVS::FAddClusterOffset>(bMainPass ?0:1);
            PermutationVectorVS.Set<FHWRasterizeVS::FMultiViewDim>(bMultiView);
            PermutationVectorVS.Set<FHWRasterizeVS::FPrimShaderDim>(bUsePrimitiveShader);
            PermutationVectorVS.Set<FHWRasterizeVS::FAutoShaderCullDim>
(bUseAutoCullingShader);
            PermutationVectorVS.Set<FHWRasterizeVS::FHasPrevDrawData>(bHavePrevDrawData);
            PermutationVectorVS.Set<FHWRasterizeVS::FVisualizeDim>
(RasterContext.VisualizeActive && Technique != ERasterTechnique::DepthOnly);
            PermutationVectorVS.Set<FHWRasterizeVS::FNearClipDim>(bNearClip);
            PermutationVectorVS.Set<FHWRasterizeVS::FVirtualTextureTargetDim>
(VirtualShadowMapArray != nullptr);
            PermutationVectorVS.Set<FHWRasterizeVS::FClusterPerPageDim>(GNaniteClusterPerPage &&
VirtualShadowMapArray != nullptr );

            //deal withPSparameter.

            FHWRasterizePS::FPermutationDomain PermutationVectorPS;
            PermutationVectorPS.Set<FHWRasterizePS::FRasterTechniqueDim>(int32(Technique));
            PermutationVectorPS.Set<FHWRasterizePS::FMultiViewDim>(bMultiView);
```

```
PermutationVectorPS.Set<FHWRasterizePS::FPrimShaderDim>(bUsePrimitiveShader);
PermutationVectorPS.Set<FHWRasterizePS::FVisualizeDim>
(RasterContext.VisualizeActive && Technique != ERasterTechnique::DepthOnly);
PermutationVectorPS.Set<FHWRasterizePS::FNearClipDim>(bNearClip);
PermutationVectorPS.Set<FHWRasterizePS::FVirtualTextureTargetDim>
(VirtualShadowMapArray != nullptr);
PermutationVectorPS.Set<FHWRasterizePS::FClusterPerPageDim>( GNaniteClusterPerPage &&
VirtualShadowMapArray != nullptr );

autoVertexShader = RasterContext.ShaderMap->GetShader<FHWRasterizeVS>
(PermutationVectorVS);
autoPixelShader = RasterContext.ShaderMap->GetShader<FHWRasterizePS>
(PermutationVectorPS);

//Adding rasterizationPass.
GraphBuilder.AddPass(
    bMainPass ? RDG_EVENT_NAME("Main Pass: Rasterize") : RDG_EVENT_NAME("Post Rasterize"),
Pass:
RasterPassParameters,
ERDGPassFlags::Raster | ERDGPassFlags::SkipRenderPass, [VertexShader,
PixelShader, RasterPassParameters, ViewRect,
bUsePrimitiveShader, bMainPass](FRHICommandListImmediate& RHICmdList)
{
    //RenderingPassinformation.FRHIRenderPassInfo RPInfo;// Resolveparameter
        .       RPInfo.ResolveParameters.DestRect.X1        =
    ViewRect.Min.X;     RPInfo.ResolveParameters.DestRect.Y1        =
    ViewRect.Min.Y;     RPInfo.ResolveParameters.DestRect.X2        =
    ViewRect.Max.X;     RPInfo.ResolveParameters.DestRect.Y2        =
    ViewRect.Max.Y;



    RHICmdList.BeginRenderPass(RPInfo, bMainPass ? TEXT("Main Pass: Rasterize") : Pass: Rasterize));
TEXT("Post
    RHICmdList.SetViewport(ViewRect.Min.X, ViewRect.Min.Y,0.0f,
FMath::Min(ViewRect.Max.X,32767), FMath::Min(ViewRect.Max.Y,32767),1.0f);

    FGraphicsPipelineStateInitializer GraphicsPSOInit;
    RHICmdList.ApplyCachedRenderTargets(GraphicsPSOInit);

    // PSO.
    GraphicsPSOInit.BlendState = TStaticBlendState<>::GetRHI();
    GraphicsPSOInit.RasterizerState =       GetStaticRasterizerState<false>(FM_Solid,
CM_CW);
    GraphicsPSOInit.DepthStencilState       = TStaticDepthStencilState<false,
CF_Always>::GetRHI();
    GraphicsPSOInit.PrimitiveType = bUsePrimitiveShader ? PT_PointList :
PT_TriangleList;
    GraphicsPSOInit.BoundShaderState.VertexDeclarationRHI       =
GEmptyVertexDeclaration.VertexDeclarationRHI;
    GraphicsPSOInit.BoundShaderState.VertexShaderRHI        =
VertexShader.GetVertexShader();
    GraphicsPSOInit.BoundShaderState.PixelShaderRHI       =
PixelShader.GetPixelShader();

    SetGraphicsPipelineState(RHICmdList, GraphicsPSOInit);

    SetShaderParameters(RHICmdList, VertexShader, VertexShader.GetVertexShader(),
RasterPassParameters->Common);
```

```
                SetShaderParameters(RHICmdList, PixelShader, PixelShader.GetPixelShader(),
* RasterPassParameters);

                RHICmdList.SetStreamSource(0, nullptr,0);
                //Note that the call isIndirecttype of interface, andIndirectArgsthat
is AddPass_InstanceHierarchyAndClusterCullResults.
                RHICmdList.DrawPrimitiveIndirect(RasterPassParameters->Common.IndirectArgs-
> GetIndirectRHICallBuffer(),16);
                RHICmdList.EndRenderPass();
            });
        }

        //Software rasterization (withCompute Shadercalculate).
        if(Scheduling != ERasterScheduling::HardwareOnly) {

            //Processing software rasterizationCSParameters. FMicropolyRasterizeCS::FPermutationDomain
            PermutationVectorCS; PermutationVectorCS.Set<FMicropolyRasterizeCS::FAddClusterOffset>(bMainPass ?0:


1);
            PermutationVectorCS.Set<FMicropolyRasterizeCS::FMultiViewDim>(bMultiView);
            PermutationVectorCS.Set<FMicropolyRasterizeCS::FHasPrevDrawData> (bHavePrevDrawData);

            PermutationVectorCS.Set<FMicropolyRasterizeCS::FRasterTechniqueDim>
            (int32(Technique));
            PermutationVectorCS.Set<FMicropolyRasterizeCS::FVisualizeDim>
(RasterContext.VisualizeActive && Technique != ERasterTechnique::DepthOnly);
            PermutationVectorCS.Set<FMicropolyRasterizeCS::FNearClipDim>(bNearClip);
            PermutationVectorCS.Set<FMicropolyRasterizeCS::FVirtualTextureTargetDim>
(VirtualShadowMapArray != nullptr);
            PermutationVectorCS.Set<FMicropolyRasterizeCS::FClusterPerPageDim>
(GNaniteClusterPerPage&& VirtualShadowMapArray != nullptr);

            autoComputeShader = RasterContext.ShaderMap->GetShader<FMicropolyRasterizeCS>
            (PermutationVectorCS);

            //  The data and p,arameters for dispatching a rasterizer call are inCommonPassParametersInside.
            FComputeShaderUtils::AddPass(
                GraphBuilder,
                bMainPass ? RDG_EVENT_NAME("Main Pass: Rasterize") : RDG_EVENT_NAME("Post Rasterize"),
Pass:
                ComputePassFlags,
                ComputeShader,
                CommonPassParameters,
                CommonPassParameters->IndirectArgs, 0);


        }
}
```

In order to further explore the process of hardware rasterization and software rasterization, it is necessary to enter their shader logic for analysis:

```
// Engine\Shaders\Private\Nanite\Rasterizer.usf

(.....)

//Rasterized triangle (for software rasterization)
```

```
 void RasterizeTri(
        FNaniteView    NaniteView,
      int4 ViewRect,
      uint PixelValue,
#if VISUALIZE
      uint2    VisualizeValues,
# endif
      float3     Verts[3],
      boolbUsePageTable        )
{
      float3 v01 = Verts[1] - Verts[0]; float3 v02 =
      Verts[2] - Verts[0];

      //Backface culling

      floatDetXY = v01.x * v02.y - v01.y * v02.x; if( DetXY >=0.0f) {


            return;
      }

      floatInvDet = rcp( DetXY ); float2 GradZ; GradZ.x = ( v01.z * v02.y - v01.y *
      v02.z ) * InvDet; GradZ.y = ( v01.x * v02.z - v01.z * v02.x ) * InvDet;



      //          8Points
                  16.Certainly  float2        Vert0       =
      Verts[0float2 Vert1 = Verts[1float2
      Vert2 = Verts[2].xy;


      //Momecnosthanpse8ag  tWaifbox  at2 MinSubpixel = min3( Vert0, Vert1, Vert2
      ) ;constfloat2 MaxSubpixel = max3( Vert0, Vert1, Vert2
      ) ;

      //Round to the nearest pixel

      int2 MinPixel = (int2)floor( (MinSubpixel + (SUBPIXEL_SAMPLES /2) -1) * (1.0/SUBPIXEL_SAMPLES) );

      int2 MaxPixel = (int2)floor((MaxSubpixel - (SUBPIXEL_SAMPLES /2) -1) * (1.0/SUBPIXEL_SAMPLES) );


      //Clip to view.

      MinPixel = max( MinPixel, ViewRect.xy ); MaxPixel =
      min( MaxPixel, ViewRect.zw -1);

      //Crop triangles with no pixel coverage

      if( any( MinPixel > MaxPixel ) )

            return;

      //Limit the rasterization bounds to a reasonable maximum value.

      MaxPixel = min( MaxPixel, MinPixel +63);

      // 4.8Fixed-point numbers

      float2 Edge01 = -v01.xy; float2 Edge12 =
      Vert1 - Vert2; float2 Edge20 = v02.xy;


      //UseMinPixelAdjustmentMinPixelPixel
      offset // 4.8 fixed point
```

```
        //Maximum triangle size =127x127Pixel
        constfloat2 BaseSubpixel = (float2)MinPixel * SUBPIXEL_SAMPLES + (SUBPIXEL_SAMPLES /
2);
        Vert0    - =  BaseSubpixel;
        Vert1    - =  BaseSubpixel;
        Vert2    - =  BaseSubpixel;

        //Ha/l/sfide8oftenquantity6 fixed pointfloatC0 = Edge01.y * Vert0.x
        -Edge01.x *Vert0.y;floatC1 = Edge12.y * Vert1.x
        -Edge12.x*Vert1.y;floatC2 = Edge20.y * Vert2.x
        -Edge20.x*Vert2.y;


        //scho/o/jlustfilTlochaprgeegulalteionbutft rule for CCWC0 -= saturate(Edge01.y +
        saturate(1.0f- Edge01.x)); C1 -= saturate(Edge12.y + saturate(1.0f-
        Edge12.x)); C2 -= saturate(Edge20.y + saturate(1.0f- Edge20.x));



        floatZ0 = Verts[0].z - ( GradZ.x * Vert0.x + GradZ.y * Vert0.y ); GradZ *=
        SUBPIXEL_SAMPLES;

        //   Calculate the step constant, andSUBPIXEL_SAMPLES The greater the Related,SUBPIXEL_SAMPLES The larger the value, the smaller the step size and the more accurate the rasterization result.
             consumption.
        floatCY0 = C0 * (1.0f/SUBPIXEL_SAMPLES); floatCY1 = C1 * (
        1.0f/SUBPIXEL_SAMPLES); floatCY2 = C2 * (1.0f/
        SUBPIXEL_SAMPLES); floatZY = Z0;


        //Whether to use scan lines
#if     COMPILER_SUPPORTS_WAVE_VOTE
        boolbScanLine = WaveActiveAnyTrue( MaxPixel.x - MinPixel.x >4);
#else
        boolbScanLine =false;
#endif

        if( bScanLine )//Scan line algorithm. {


            float3 Edge012 = { Edge01.y, Edge12.y, Edge20.y }; bool3
            bOpenEdge = Edge012 <0;
            float3 InvEdge012 = Edge012 ==0?1e8: rcp( Edge012 );

            inty = MinPixel.y; while(
            true)
            {
                // No longer fixed point
                float3 CrossX = float3( CY0, CY1, CY2 ) * InvEdge012;

                float3 MinX = bOpenEdge ? CrossX :0;
                float3 MaxX = bOpenEdge ? MaxPixel.x - MinPixel.x : CrossX;

                floatx0 =ceil( max3( MinX.x, MinX.y, MinX.z ) ); floatx1 =
                min3( MaxX.x, MaxX.y, MaxX.z ); floatZX = ZY + GradZ.x * x0;


                x0   + =  MinPixel.x;
                x1   + =  MinPixel.x;
                //      Traverse all pixels in a direction and write pixel data.
                        x
```

```
                for(floatx = x0; x <= x1; x++ ) {

                        //Write pixel values and depth values.
                        WritePixel(OutVisBuffer64, PixelValue, uint2(x,y), ZX, NaniteView,
bUsePageTable);
                #if    VISUALIZE
                        WritePixel(OutDbgBuffer64, VisualizeValues.x, uint2(x,y), ZX, NaniteView,
bUsePageTable);
                        InterlockedAdd(OutDbgBuffer32[uint2(x,y)], VisualizeValues.y);
                #endif

                        ZX += GradZ.x;
                }

                if( y >= MaxPixel.y )
                        break;

                //IncreaseYC 0 FangYStep towards + =
                Edge01.x;      CY1   + =
                Edge12.x;      CY2   + =
                Edge20.x;       ZY   + =
                GradZ.y;      y++;

        }
    }
    else//  Non-scanning line algorithm (Rectangular frame algorithm, need to detect whether it is inside the triangle)
    {
        inty = MinPixel.y;

        while(true)
        {
            intx = MinPixel.x;
            // 3All of them are positive numbers, which means they are
            inside the triangle. if(min3(CY0, CY1, CY2) >=0) {

                WritePixel(OutVisBuffer64, PixelValue, uint2(x, y), ZY, NaniteView,
bUsePageTable);
            #ifVISUALIZE
                WritePixel(OutDbgBuffer64, VisualizeValues.x, uint2(x, y), ZY, NaniteView,
bUsePageTable);
                InterlockedAdd(OutDbgBuffer32[uint2(x, y)], VisualizeValues.y);
            #endif
            }

            if(x < MaxPixel.x) {

                floatCX0 = CY0 - Edge01.y; floatCX1 =
                CY1 - Edge12.y; floatCX2 = CY2 -
                Edge20.y; floatZX = ZY + GradZ.x; x++;



                HOIST_DESCRIPTORS
                while(true)
                {
                    if(min3(CX0, CX1, CX2) >=0) {

                        WritePixel(OutVisBuffer64, PixelValue, uint2(x, y), ZX,
```

```
NaniteView,      bUsePageTable);
                        #if VISUALIZE
                            WritePixel(OutDbgBuffer64, VisualizeValues.x, uint2(x, y), ZX,
NaniteView,      bUsePageTable);
                            InterlockedAdd(OutDbgBuffer32[uint2(x, y)], VisualizeValues.y);
                        #endif
                        }

                        if(x >= MaxPixel.x)
                            break;

                        CX0  -  =  Edge01.y;
                        CX1  -  =  Edge12.y;
                        CX2  -  =  Edge20.y;
                        ZX    + =   GradZ.x;
                        x++;
                    }
                }

                if(y >= MaxPixel.y)
                    break;

                CY0  + =  Edge01.x;
                CY1  + =  Edge12.x;
                CY2  + =  Edge20.x;
                ZY    + =   GradZ.y;
                y++;
            }
        }
}

#ifUSE_CONSTRAINED_CLUSTERS
  groupshared float3       GroupVerts[256];
#else
groupshared     float3      GroupVerts[384];
#endif

//Detect cropping mode, the mode is clockwise (CW).If it returnstrue,Need to rotate counterclockwise (CCW).
bool ReverseWindingOrder(FInstanceSceneData InstanceData) {

    boolbReverseInstanceCull = (InstanceData.InvNonUniformScaleAndDeterminantSign.w < 0.0f);

    boolbRasterStateReverseCull = (RenderFlags & RENDER_FLAG_REVERSE_CULLING);

    // Logical XOR
    return(bReverseInstanceCull != bRasterStateReverseCull);
}

StructuredBuffer< uint2 >                   InTotalPrevDrawClusters;
Buffer<uint>                                InClusterOffsetSWHW;

groupsshared float4x4 LocalToSubpixelLDS;

//Microsurface rasterization, used for      NaniteofCSSoft grating.
[numthreads(128,        1,1)]
void MicropolyRasterize(
    uint        VisibleIndex            : SV_GroupID,
    uint        GroupIndex                : SV_GroupIndex)
```

```
{
        //Compute visible indices.
#ifHAS_PREV_DRAW_DATA
    VisibleIndex      + = InTotalPrevDrawClusters[0].x;
#endif
#if   ADD_CLUSTER_OFFSET
    VisibleIndex += InClusterOffsetSWHW[0];
#endif

    //Get visibleClusterand instance data.
    FVisibleCluster VisibleCluster = GetVisibleCluster( VisibleIndex,
    VIRTUAL_TEXTURE_TARGET );
    FInstanceSceneData InstanceData = GetInstanceData( VisibleCluster.InstanceId ); //GetNaniteview.

    FNaniteView NaniteView = GetNaniteView( VisibleCluster.ViewId );

    //Get page information.
#if CLUSTER_PER_PAGE
    // Scalar
    uint2 vPage = VisibleCluster.vPage;
    FShadowPhysicalPage pPage =
ShadowGetPhysicalPage( CalcPageTableLevelOffset( NaniteView.TargetLayerIndex,
NaniteView.TargetMipLevel ) + CalcPageOffsetInLevel( NaniteView.TargetMipLevel, vPage ) );
#endif

    float4x4 LocalToSubpixel;

    // InstancedDynamicDatayesGroupIt is constant, so it only needs to be calculated once and then stored in      groupsharedvariables for subsequent use
Use.
    if( GroupIndex ==0) {

        LocalToSubpixel = CalculateInstanceDynamicData(NaniteView,
        InstanceData).LocalToClip;

        float2 Scale = float2(0.5,-0.5) * NaniteView.ViewSizeAndInvSize.xy * SUBPIXEL_SAMPLES;

        float2 Bias = (0.5* NaniteView.ViewSizeAndInvSize.xy + NaniteView.ViewRect.xy )
*SUBPIXEL_SAMPLES +0.5f;

#ifCLUSTER_PER_PAGE
        Bias += ( (float2)pPage.PageIndex - (float2)vPage ) * VSM_PAGE_SIZE * SUBPIXEL_SAMPLES;

#endif

        LocalToSubpixel._m00_m10_m20_m30 = LocalToSubpixel._m00_m10_m20_m30 * Scale.x +
LocalToSubpixel._m03_m13_m23_m33 * Bias.x;
        LocalToSubpixel._m01_m11_m21_m31 = LocalToSubpixel._m01_m11_m21_m31 * Scale.y +
LocalToSubpixel._m03_m13_m23_m33 * Bias.y;

        LocalToSubpixelLDS = LocalToSubpixel;
    }
    //UsageGroupMemory barriers for synchronization
    Groupdata. GroupMemoryBarrierWithGroupSync();
    LocalToSubpixel = LocalToSubpixelLDS;


    //GetClusterdata.
    FCluster Cluster = GetCluster(VisibleCluster.PageIndex, VisibleCluster.ClusterIndex);
```

```
        UNROLL
        for( uint i =0; i <2; i++ ) {

                uint VertIndex = GroupIndex + i *128; if(VertIndex <
                Cluster.NumVerts) {

                        //Transform vertices and save them to shared memory between groups.
                        float3 PointLocal = DecodePosition( VertIndex, Cluster );
                        float4 PointClipSubpixel = mul( float4( PointLocal,1), LocalToSubpixel ); float3 Subpixel =
                        PointClipSubpixel.xyz / PointClipSubpixel.w;
                        GroupVerts[VertIndex] = float3(floor(Subpixel.xy), Subpixel.z);

                }
        }
        //UsageGroupMemory barriers for synchronization
        Groupdata. GroupMemoryBarrierWithGroupSync();


        int4 ViewRect = NaniteView.ViewRect;

#if     CLUSTER_PER_PAGE
        ViewRect.xy = pPage.PageIndex * VSM_PAGE_SIZE;
        ViewRect.zw = ViewRect.xy + VSM_PAGE_SIZE;
#endif

        if(GroupIndex < Cluster.NumTris) {

                //TriangleIDthat isGroupindex. uint TriangleID =
                GroupIndex; //Generates a triangular index, and handles
                the case where flipping is required.
                uint3 TriangleIndices = ReadTriangleIndices(Cluster, TriangleID); if
                (ReverseWindingOrder(InstanceData)) {


                        TriangleIndices = uint3(TriangleIndices.x, TriangleIndices.z,
TriangleIndices.y);
                }

                //Get threeCornerlocation. float3 Vertices[3]; Vertices0]        =
                GroupVerts[TriangleIndices.x];                Vertices1]         =
                GroupVerts[TriangleIndices.y];                Vertices2]         =
                GroupVerts[TriangleIndices.z];


                //The pixel value is the triangleID.

                uint PixelValue = ((VisibleIndex +1) <<7) | TriangleID;

                //Rasterize the triangle and write the corresponding sumideep append .

                RasterizeTri(
                        NaniteView,
                        ViewRect,
                        PixelValue,
                #if VISUALIZE
                        GetVisualizeValues(),
                #endif
                        Vertices,
                        !CLUSTER_PER_PAGE );
        }
}
```

```
#define PIXEL_VALUE (RASTER_TECHNIQUE != RASTER_TECHNIQUE_DEPTHONLY)
#define VERTEX_TO_TRIANGLE_MASKS (NANITE_PRIM_SHADER && PIXEL_VALUE)

structVSOut
{
    noperspective float DeviceZ : TEXCOORD0;
#ifPIXEL_VALUE
    nointerpolation uint PixelValue : TEXCOORD1;
#endif
#if NANITE_MULTI_VIEW
    nointerpolation int4 ViewRect : TEXCOORD2;
#endif
#if VISUALIZE
    nointerpolation uint2 VisualizeValues : TEXCOORD3;
#endif
#if VIRTUAL_TEXTURE_TARGET
    nointerpolation intViewId : TEXCOORD4;
#endif
#if VERTEX_TO_TRIANGLE_MASKS
    CUSTOM_INTERPOLATION uint4 ToTriangleMasks : TEXCOORD5;
#endif
    float4 Position : SV_Position;
};

//Hardware RasterizationVS,Mainly, vertex data is converted fromClusterDecompress it and transform it to clip space.
VSOutCommonRasterizerVS(FNaniteView NaniteView, FInstanceSceneData InstanceData, FVisibleCluster
VisibleCluster, FCluster Cluster, uint VertIndex, out float4 PointClipNoScaling)

{
    VSOut Out;

    float4x4 LocalToWorld = InstanceData.LocalToWorld;

    float3 PointLocal = DecodePosition( VertIndex, Cluster );
    float3 PointRotated = LocalToWorld[0].xyz * PointLocal.xxx + LocalToWorld[1].xyz * PointLocal.yyy +
LocalToWorld[2].xyz * PointLocal.zzz;
    float3 PointTranslatedWorld = PointRotated + (LocalToWorld[3].xyz +
    NaniteView.PreViewTranslation.xyz);
    float4 PointClip = mul( float4( PointTranslatedWorld,1),
    NaniteView.TranslatedWorldToClip );
    PointClipNoScaling = PointClip;
#if CLUSTER_PER_PAGE
    PointClip.xy = NaniteView.ClipSpaceScaleOffset.xy * PointClip.xy +
NaniteView.ClipSpaceScaleOffset.zw * PointClip.w;

    // Offset 0,0 to be at vPage for a 0, VSM_PAGE_SIZE * VSM_RASTER_WINDOW_PAGES viewport.

    PointClip.xy += PointClip.w * ( float2(-2,2) / VSM_RASTER_WINDOW_PAGES ) * VisibleCluster.vPage;


    Out.ViewRect.xy = VisibleCluster.vPage * VSM_PAGE_SIZE; Out.ViewRect.zw
    = NaniteView.ViewRect.zw;
#elifNANITE_MULTI_VIEW
    PointClip.xy = NaniteView.ClipSpaceScaleOffset.xy * PointClip.xy +
NaniteView.ClipSpaceScaleOffset.zw * PointClip.w;
    Out.ViewRect = NaniteView.ViewRect;
#endif
```

```
#if    VIRTUAL_TEXTURE_TARGET
        Out.ViewId =        VisibleCluster.ViewId;
#endif
        Out.Position        =PointClip;
        Out.DeviceZ = PointClip.z / PointClip.w;

        // Shader workaround to avoid HW depth clipping. Should be replaced with rasterizer state
            ideally.
#if    !NEAR_CLIP
        Out.Position.z =0.5f* Out.Position.w;
#endif

#if    VISUALIZE
        Out.VisualizeValues          =  GetVisualizeValues();
#endif
        returnOut;
}

#ifNANITE_PRIM_SHADER

#Pragmas  argument(wavemode=wave64)
#Pragmas  argument(realtypes)

structPrimitiveInput
{
        uint    Index               :    PRIM_SHADER_SEM_VERT_INDEX;
        uint    WaveIndex           :    PRIM_SHADER_SEM_WAVE_INDEX;
};

structPrimitiveOutput
{
        VSOut Out;

        uint      PrimExport        :   PRIM_SHADER_SEM_PRIM_EXPORT;
        uint        VertCount       :    PRIM_SHADER_SEM_VERT_COUNT;
        uint    PrimCount           :    PRIM_SHADER_SEM_PRIM_COUNT;
};
//Compressed triangle index, wherex,y,zThe number of digits is
10,10,12. uintPackTriangleExport(uint3 TriangleIndices) {


        returnTriangleIndices.x | (TriangleIndices.y <<10) | (TriangleIndices.z <<20);
}

//Unpack the triangle index.
uint3UnpackTriangleExport(uint Packed) {

        =co (nPsatcukiendt I>n>d1e0x)0 & =0 x(P3aFcFk;/e/dEx &tr0axc3tF tFh)e;/ /mBeidfdolree1 e0xBtirtaction10Bit. constuint Index1

        constuint Index2 = (Packed >>20);//After extraction12Bit. return
        uint3(Index0, Index1, Index2);

}

#ifVERTEX_TO_TRIANGLE_MASKS//Triangle mask rendering mode.
groupshared     uint    GroupVertexToTriangleMasks[256][4];
#endif
groupshared     uint    GroupTriangleCount;
groupshared     uint    GroupVertexCount;
```

```
groupsshared uint GroupClusterIndex;

PRIM_SHADER_OUTPUT_TRIANGLES
PRIM_SHADER_PRIM_COUNT(1)
PRIM_SHADER_VERT_COUNT(1)
PRIM_SHADER_VERT_LIMIT(256)
PRIM_SHADER_AMP_FACTOR(128)
PRIM_SHADER_AMP_ENABLE

//Hardware RasterizationVEnterSOCTET 3 (rectangular mask rendering mode)                ) .
PrimitiveOutputHWRasterizeVS(PrimitiveInput Input) {

    constuint LaneIndex = WaveGetLaneIndex(); constuint
    LaneCount = WaveGetLaneCount();

    constuint GroupThreadID = LaneIndex + Input.WaveIndex * LaneCount;

    if(GroupThreadID ==0) {

        // Input index is only initialized for lane 0, so we need to manually communicate it to all other threads in
subgroup (not just wavefront).
        GroupClusterIndex = Input.Index;
    }

    GroupMemoryBarrierWithGroupSync();

    //The following code andMicropolyRasterizeType, omitted        .
    uint VisibleIndex = GroupClusterIndex;
#if  HAS_PREV_DRAW_DATA
    VisibleIndex       + = InTotalPrevDrawClusters[0].y;
#endif
#if  ADD_CLUSTER_OFFSET
    VisibleIndex += InClusterOffsetSWHW[GetHWClusterCounterIndex(RenderFlags)];
#endif
    VisibleIndex = (MaxVisibleClusters -1) - VisibleIndex;

    // Should be all scalar.
    FVisibleCluster VisibleCluster = GetVisibleCluster( VisibleIndex,
    VIRTUAL_TEXTURE_TARGET );
    FInstanceSceneData InstanceData = GetInstanceData( VisibleCluster.InstanceId ); FNaniteView NaniteView
    = GetNaniteView( VisibleCluster.ViewId );

    FInstanceDynamicData InstanceDynamicData = CalculateInstanceDynamicData(NaniteView, InstanceData);


    FCluster Cluster = GetCluster(VisibleCluster.PageIndex, VisibleCluster.ClusterIndex);

#if  VERTEX_TO_TRIANGLE_MASKS
    if(GroupThreadID < Cluster.NumVerts) {

        GroupVertexToTriangleMasks[GroupThreadID][0]          =  0;
        GroupVertexToTriangleMasks[GroupThreadID][1]          =  0;
        GroupVertexToTriangleMasks[GroupThreadID][2]          =  0;
        GroupVertexToTriangleMasks[GroupThreadID][3]          = 0;
    }
#endif

    GroupMemoryBarrierWithGroupSync();
```

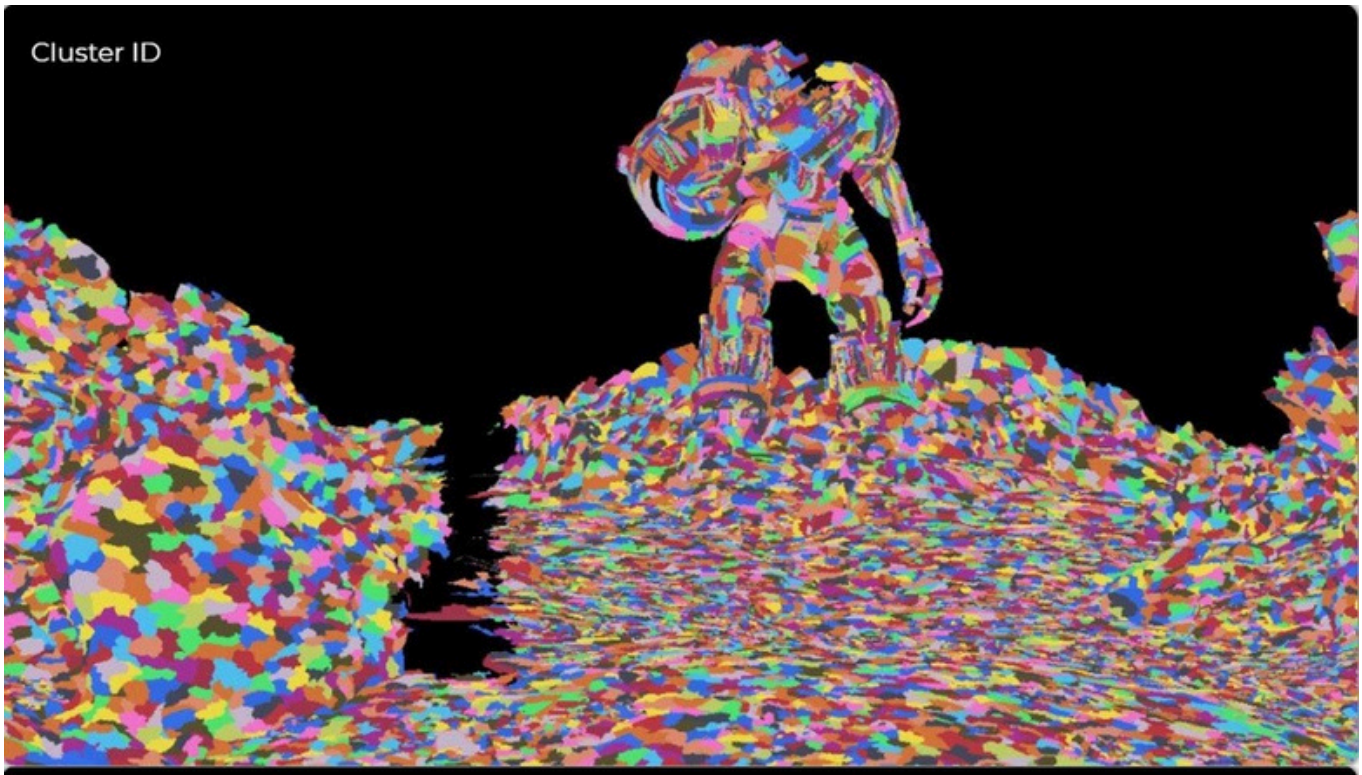
```
        PrimitiveOutput PrimOutput;
        PrimOutput.VertCount        = Cluster.NumVerts;
        PrimOutput.PrimCount        = Cluster.NumTris;

        boolbCullTriangle =false;

        if(GroupThreadID < Cluster.NumTris) {

                uint TriangleID = GroupThreadID;
                uint3 TriangleIndices = ReadTriangleIndices(Cluster, TriangleID); if
                (ReverseWindingOrder(InstanceData)) {

                        TriangleIndices = uint3(TriangleIndices.x, TriangleIndices.z,
TriangleIndices.y);
                }

#ifVERTEX_TO_TRIANGLE_MASKS const
                 uint DwordIndex                    = (GroupThreadID >>5) &3;
                constuint TriangleMask =1<< (GroupThreadID &31);
                InterlockedOr(GroupVertexToTriangleMasks[TriangleIndices.x][DwordIndex], TriangleMask);

                InterlockedOr(GroupVertexToTriangleMasks[TriangleIndices.y][DwordIndex], TriangleMask);

                InterlockedOr(GroupVertexToTriangleMasks[TriangleIndices.z][DwordIndex], TriangleMask);

#endif
                PrimOutput.PrimExport = PackTriangleExport(TriangleIndices);
        }

        GroupMemoryBarrierWithGroupSync();

        if(GroupThreadID < Cluster.NumVerts) {

                float4 PointClipNoScaling;
                //Rasterize triangles.
                PrimOutput.Out = CommonRasterizerVS(NaniteView, InstanceData, VisibleCluster, Cluster,
GroupThreadID, PointClipNoScaling);
#ifVERTEX_TO_TRIANGLE_MASKS
                PrimOutput.Out.PixelValue = ((VisibleIndex +1) <<7); PrimOutput.Out.ToTriangleMasks =
                uint4(GroupVertexToTriangleMasks[GroupThreadID]
[0],
                                                                GroupVertexToTriangleMasks[GroupThreadID]
[1],
                                                                GroupVertexToTriangleMasks[GroupThreadID]
[2],
                                                                GroupVertexToTriangleMasks[GroupThreadID]
[3]);
#endif
        }

        returnPrimOutput;
}

#else// NANITE_PRIM_SHADER(NanitePrimitive coloring mode)

//Hardware RasterizationVEnterSImage (meta coloring mode          ) .
VSOutHWRasterizeVS(
```

```
        uint    VertexID                :   SV_VertexID,
        uint    VisibleIndex            :   SV_InstanceID
        )
{                       #   if
HAS_PREV_DRVAisW
            ibl_iDndAexT+=AInTotalPrevDrawClusters[0].y;
#endif

#if    ADD_CLUSTER_OFFSET
        VisibleIndex += InClusterOffsetSWHW[GetHWClusterCounterIndex(RenderFlags)];
#endif
        VisibleIndex = (MaxVisibleClusters -1) - VisibleIndex;

        uint TriIndex = VertexID /3; VertexID = VertexID
        - TriIndex *3;

        VSOut Out;
        Out.Position        =   float4(0,0,0,1); 0.0f;
        Out.DeviceZ      =

        FVisibleCluster VisibleCluster = GetVisibleCluster( VisibleIndex,
        VIRTUAL_TEXTURE_TARGET );
        FInstanceSceneData InstanceData = GetInstanceData( VisibleCluster.InstanceId );

        FNaniteView NaniteView = GetNaniteView( VisibleCluster.ViewId );
        FCluster Cluster = GetCluster(VisibleCluster.PageIndex, VisibleCluster.ClusterIndex);

        if(TriIndex < Cluster.NumTris) {

            uint3 TriangleIndices = ReadTriangleIndices( Cluster, TriIndex ); if
            (ReverseWindingOrder(InstanceData)) {

                TriangleIndices = uint3( TriangleIndices.x, TriangleIndices.z,
TriangleIndices.y );
            }

            uint VertIndex = TriangleIndices[VertexID]; float4
            PointClipNoScaling;
            //Rasterize triangles.
            Out = CommonRasterizerVS(NaniteView, InstanceData, VisibleCluster, Cluster, VertIndex,

                PointClipNoScaling);
        #if    PIXEL_VALUE
            Out.PixelValue = ((VisibleIndex +1) <<7) | TriIndex;
        #endif
        }

        returnOut;
}

#endif// NANITE_PRIM_SHADER

//Hardware RasterizationPEnter                sMouth .
void HWRasterizePS(VSOut In) {

        uint2 PixelPos = (uint2)In.Position.xy;

        uint PixelValue =0;
#if    PIXEL_VALUE
```

```
        PixelValue = In.PixelValue;
#endif

#if    VERTEX_TO_TRIANGLE_MASKS
        uint4 Masks0 = LoadParameterCacheP0( In.ToTriangleMasks ); uint4 Masks1 =
        LoadParameterCacheP1( In.ToTriangleMasks ); uint4 Masks2 =
        LoadParameterCacheP2( In.ToTriangleMasks );

        uint4 Masks = Masks0 & Masks1 & Masks2; uint
        TriangleIndex =                 Masks.x ? firstbitlow( Masks.x ) : Masks.y ?
                                        firstbitlow( Masks.y ) +32: Masks.z ?
                                        firstbitlow( Masks.z ) +64: firstbitlow( Masks.w ) +96;


        PixelValue += TriangleIndex;
#endif

#if    VIRTUAL_TEXTURE_TARGET
        FNaniteView NaniteView = GetNaniteView(In.ViewId);
#else
        FNaniteView NaniteView;
#endif

#if    CLUSTER_PER_PAGE
        PixelPos += In.ViewRect.xy; if(all(PixelPos <
        In.ViewRect.zw))
#elifNANITE_MULTI_VIEW
        // In multi-view mode every view has its own scissor, so we have to scissor manually. if(all(PixelPos >=
        In.ViewRect.xy && PixelPos < In.ViewRect.zw))
#endif
        {
                //Writing pixel data: triangleid(PixelValue),depth(In.DeviceZ) WritePixel(OutVisBuffer64,
                PixelValue, PixelPos, In.DeviceZ, NaniteView,

VIRTUAL_TEXTURE_TARGET);
        #if VISUALIZE
                WritePixel(OutDbgBuffer64, In.VisualizeValues.x, PixelPos, In.DeviceZ, NaniteView, VIRTUAL_TEXTURE_TARGET);

                InterlockedAdd(OutDbgBuffer32[PixelPos], In.VisualizeValues.y);
        #endif
        }
}
```

From the above analysis, we can see that no matter it is software raster or hardware raster, the only thing that writes data isClusterID, triangleIDAnd depth (if it is in visualization mode, there are other data), which means that this stage is not really colored, but similar to delayed rendering.BasePass, but the output information is far fromBasePassThe resultingIO, video memory are significantly reduced. In fact, this technology isVisibility BufferTechnology, please refer toAnalysis of Unreal Rendering System (04) - Deferred Rendering PipelineDetails4.2.3.5 Visibility Buffer.

The storage structure of Nanite after rasterization: ClusterID occupies 25 bits, triangle ID occupies 7 bits, and depth occupies 32 bits.

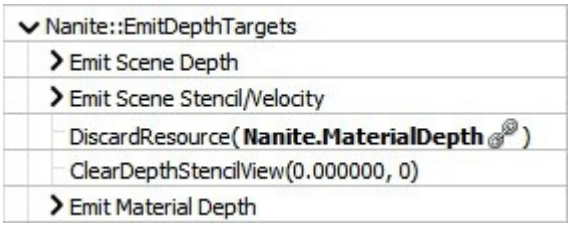

Cluster ID


Triangle ID


Depth

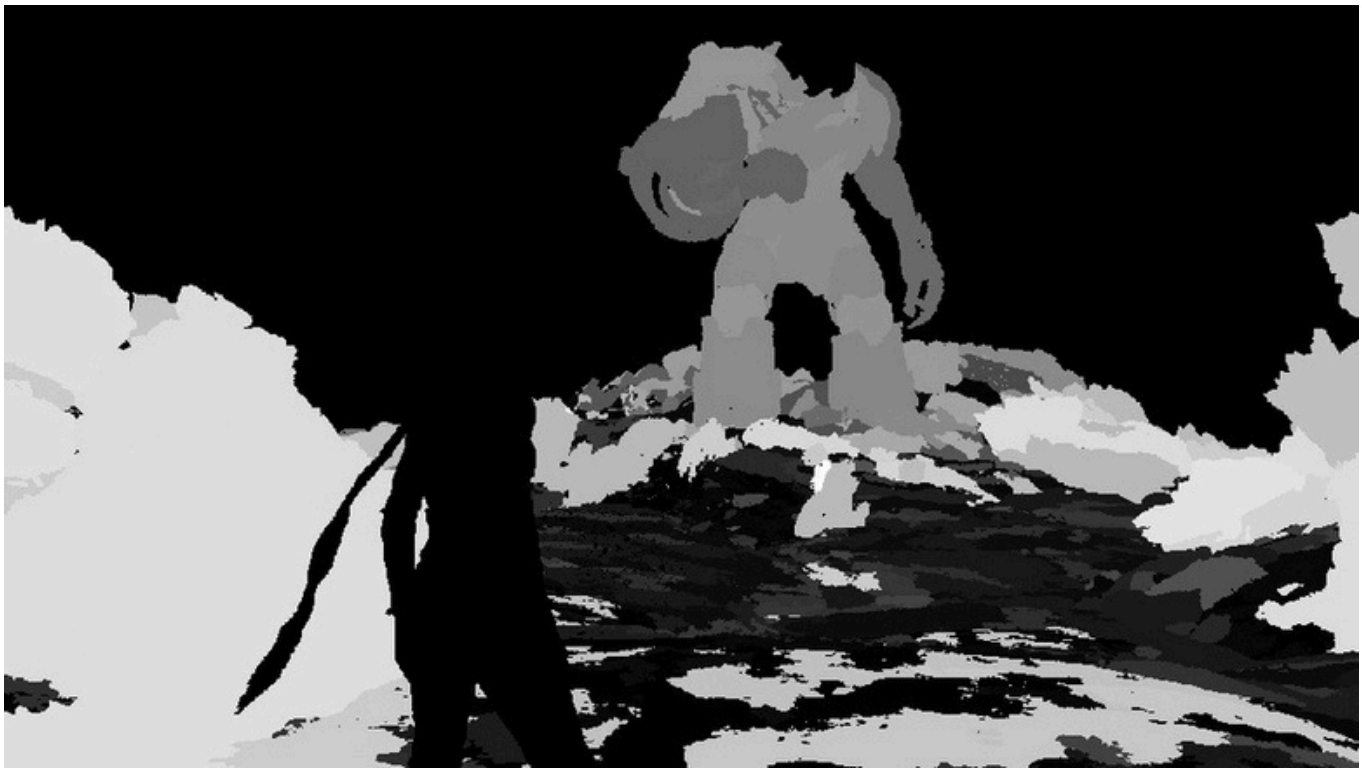
Schematic diagram of the result after Nanite rasterization, from top to bottom are ClusterID, triangle ID, and depth.

After Nanite rasterization, another important step is**Nanite::EmitDepthTargets**,which is used to buffer the scene's depth, template, speed, material depth, etc.:



The template buffer indicates which pixels are rendered by Nanite:

The most interesting thing is the material depth, which indicates which material is covered by each pixel at the front of the scene. It is essentially a material ID converted to a unique depth value and stored in the depth template texture. In fact, each material has a grayscale value for subsequent optimization using Early Z.



6.4.3.6 Nanite BasePass

This section mainly describes how Nanite's BasePass generates GBuffer. The main process in FDeferredShadingSceneRenderer::Render is as follows:

```
void FDeferredShadingSceneRenderer::Render(FRDGBuilder& GraphBuilder) {

    (.....)
    //RenderingNaniteofBasePass. {


        //Draw normal modeBasePass.
        RenderBasePass(GraphBuilder,        SceneTextures,      DBufferTextures,
BasePassDepthStencilAccess, ForwardScreenSpaceShadowMaskTexture,                InstanceCullingManager);
        AddServiceLocalQueuePass(GraphBuilder);

        if(bNaniteEnabled && bShouldApplyNaniteMaterials) {

            for(int32 ViewIndex =0; ViewIndex < Views.Num(); ++ViewIndex) {

                constFViewInfo& View = Views[ViewIndex];
                Nanite::FRasterResults& RasterResults = NaniteRasterResults[ViewIndex];

                //  If depth was not drawn earlier, draw it now
                if  (!bNeedsPrePass)
                {
                    Nanite::EmitDepthTargets(
                        GraphBuilder,
                        * Scene,
                        Views[ViewIndex],
                        RasterResults.SOAStrides,
                        RasterResults.VisibleClustersSWHW,
                        RasterResults.ViewsBuffer,
                        SceneTextures.Depth.Target,
                        RasterResults.VisBuffer64,
```

```
                        RasterResults.MaterialDepth,
                        RasterResults.NaniteMask,
                        RasterResults.VelocityBuffer,
                        bNeedsPrePass
                    );
                }

                //drawNanitePatternBasePass.
                Nanite::DrawBasePass(
                GraphBuilder,
                SceneTextures,
                DBufferTextures,
                * Scene,
                View,
                RasterResults
                );
            }
        }

        // Resolve scene depth.
        if (!bAllowReadOnlyDepthBasePass)
        {
            AddResolveSceneDepthPass(GraphBuilder, Views, SceneTextures.Depth);
        }

        (......)
    }

    (......)
}
```

It should be noted that there are two BasePass renderings above: one is the traditional BasePass rendering RenderBasePass, and the other is the Nanite mode BasePass rendering Nanite::DrawBasePass. The following is the analysis of Nanite::DrawBasePass:

```
// Engine\Source\Runtime\Renderer\Private\Nanite\NaniteRender.cpp

void DrawBasePass(
    FRDGBuilder&     GraphBuilder,
    const    FSceneTextures& SceneTextures,
    const    FDBufferTextures& DBufferTextures, FScene&
    const    Scene,
    const    FViewInfo& View,
    const    FRasterResults & RasterResults
)
{
    (......)

    RDG_EVENT_SCOPE(GraphBuilder,"Nanite::BasePass");

    constint32 ViewWidth                    = View.ViewRect.Max.X - View.ViewRect.Min.X; =
    constint32 ViewHeight                     View.ViewRect.Max.Y - View.ViewRect.Min.Y;
    constFIntPoint ViewSize                 =FIntPoint(ViewWidth, ViewHeight);

    constFRDGSystemTextures& SystemTextures = FRDGSystemTextures::Get(GraphBuilder);
```

```
	FRenderTargetBindingSlots GBufferRenderTargets;
	SceneTextures.GetGBufferRenderTargets(ERenderTargetLoadAction::ELoad,
	GBufferRenderTargets);

	//Initializes a texture reference.

	FRDGTextureRef MaterialDepth RasterResul=t sR.MasatteerrRiaelsDueltpst.Mh :a ter?alDepth
	SystemTextures.Black;
	FRDGTextureRef VisBuffer64 =RasterResults.VisBuffer64 ?
RasterResults.VisBuffer64 : SystemTextures.Black;
	FRDGTextureRef DbgBuffer64 =RasterResults.DbgBuffer64 ?
RasterResults.DbgBuffer64 : SystemTextures.Black;
	FRDGTextureRef DbgBuffer32 =RasterResults.DbgBuffer32 ?
RasterResults.DbgBuffer32 : SystemTextures.Black;

	FRDGBufferRef VisibleClustersSWHW = RasterResults.VisibleClustersSWHW;

	// Detect material clipping mode. Wave operation requiresSM6Only supported platforms will be switched to4. (!
	if FDataDrivenShaderPlatformInfo::GetSupportsWaveOperations(GMaxRHIShaderPlatform)
&&
		(GNaniteMaterialCulling ==1 || GNaniteMaterialCulling==2))
	{
		UE_LOG(LogNanite, Warning, TEXT("r.Nanite.MaterialCulling set to %d which requires wave-ops (not supported
on this platform), switching to mode 4"), GNaniteMaterialCulling);
		GNaniteMaterialCulling=4;
	}

	//By using local assignment, you can achieve the purpose of coverage without modifying all views.

	int32 NaniteMaterialCulling = GNaniteMaterialCulling;
	if((NaniteMaterialCulling ==1 || NaniteMaterialCulling==2) && (View.ViewRect.Min.X !=0 || View.ViewRect.Min.Y !=0))

	{
		NaniteMaterialCulling=4;

		static boolbLoggedAlready =false; if(!
		bLoggedAlready)
		{
			bLoggedAlready =true;
			UE_LOG(LogNanite, Warning, TEXT("View has non-zero viewport offset, using
material culling mode 4 (overrides r.Nanite.MaterialCulling = %d)."), GNaniteMaterialCulling);

		}
	}

	//Bitmask Clipping

	const boolb32BitMaskCulling = (NaniteMaterialCulling ==1 || NaniteMaterialCulling==
2);
	//Block cutting
	const boolbTileGridCulling = (NaniteMaterialCulling ==3 || NaniteMaterialCulling==

4);

	constFIntPoint TileGridDim = bTileGridCulling ? FMath::DivideAndRoundUp(ViewSize, { 64,64}) : FIntPoint(1,1);


	//Create textures and buffers.

	FRDGBufferDesc VisibleMaterialsDesc = FRDGBufferDesc::CreateStructuredDesc(4,
b32BitMaskCulling ? FNaniteCommandInfo::MAX_STATE_BUCKET_ID+1:1);
	FRDGBufferRef VisibleMaterials =
GraphBuilder.CreateBuffer(VisibleMaterialsDesc, TEXT("Nanite.VisibleMaterials"));
```

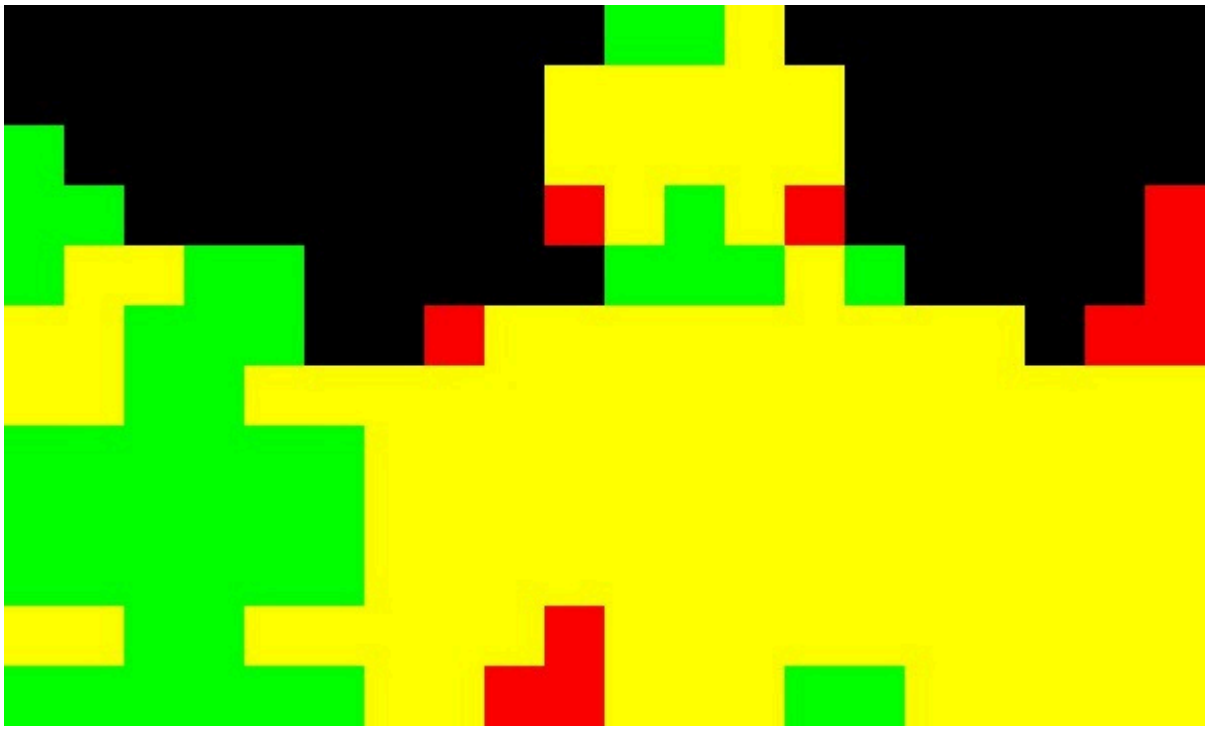
```
        FRDGBufferUAVRef        VisibleMaterialsUAV             =GraphBuilder.CreateUAV(VisibleMaterials);
        FRDGTextureDesc         MaterialRangeDesc               =FRDGTextureDesc::Create2D(TileGridDim,
    PF_R32G32_UINT, FClearValueBinding::Black, TexCreate_ShaderResource | TexCreate_UAV);
        FRDGTextureRef          MaterialRange                   =
GraphBuilder.CreateTexture(MaterialRangeDesc,                   TEXT("Nanite.MaterialRange"));
        FRDGTextureUAVRef       MaterialRangeUAV                = GraphBuilder.CreateUAV(MaterialRange);
        FRDGTextureSRVDesc      MaterialRangeSRVDesc            =
FRDGTextureSRVDesc::Create(MaterialRange);
        FRDGTextureSRVRef MaterialRangeSRV                      =
        GraphBuilder.CreateSRV(MaterialRangeSRVDesc);

        //Cleaning up the texture buffer

        AddClearUAVPass(GraphBuilder, VisibleMaterialsUAV,0); AddClearUAVPass(GraphBuilder,
        MaterialRangeUAV, {0u,1u,0u,0u});

        //Categorize materials to cut in blocks

        if(b32BitMaskCulling || bTileGridCulling) {

                FClassifyMaterialsCS::FParameters* PassParameters                       =
GraphBuilder.AllocParameters<FClassifyMaterialsCS::FParameters>(); PassParameters-
>View                                   PassParameters-         = View.ViewUniformBuffer;
> VisibleClustersSWHW                                           =
GraphBuilder.CreateSRV(VisibleClustersSWHW);
                PassParameters->SOAStrides PassParameters-              =RasterResults.SOAStrides; =
                >ClusterPageData
                Nanite::GStreamingManager.GetClusterPageDataSRV();
                PassParameters->ClusterPageHeaders                      =
                Nanite::GStreamingManager.GetClusterPageHeadersSRV();
                PassParameters->VisBuffer64 PassParameters->MaterialDe=ptVhiTsBabulfefe r64;
                Scene.MaterialTables[ENaniteMeshPass::BasePass].GetDe=pthTableSRV();


                uint32 DispatchGroupSize =0;

                PassParameters->ViewRect = FIntVector4(View.ViewRect.Min.X, View.ViewRect.Min.Y, View.ViewRect.Max.X,
                View.ViewRect.Max.Y);
                if(b32BitMaskCulling)
                {
                        checkf(View.ViewRect.Min.X ==0&& View.ViewRect.Min.Y ==0, TEXT("Viewport
offset support is not implemented."));
                        DispatchGroupSize =8; PassParameters->VisibleMaterials =
                        VisibleMaterialsUAV;

                }
                else            if          ( b Tile G rid C ulling )
                {
                        DispatchGroupSize       =64;
                        PassParameters->FetchClamp = View.ViewRect.Max -1;
                        PassParameters->MaterialRange = MaterialRangeUAV;
                }

                constFIntVector DispatchDim =
FComputeShaderUtils::GetGroupCount(View.ViewRect.Max                    -   View.ViewRect.Min,
DispatchGroupSize);

                FClassifyMaterialsCS::FPermutationDomain PermutationVector;
                PermutationVector.Set<FClassifyMaterialsCS::FCullingMethodDim>
                (NaniteMaterialCulling);
```

```
        autoComputeShader = View.ShaderMap->GetShader<FClassifyMaterialsCS>
        (PermutationVector.ToDimensionValueId());

        //Classification of materialsCS Pass.
        FComputeShaderUtils::AddPass(
                GraphBuilder,
                RDG_EVENT_NAME("Classify        Materials"),
                ComputeShader,
                PassParameters,
                DispatchDim
        ) ;
}

//RenderingGBuffer.
{
        //      deal withPassdata
        FNaniteEmitGBufferParameters*           PassParameters      =
        GraphBuilder.AllocParameters<FNaniteEmitGBufferParameters>();

        PassParameters->SOAStrides PassParam=e tRears-t>eMrRaexsVuilstisb.SleOCAluSstrteidres s;
        PassParameters->MaxNodes = RasterResults.MaxNo=d Reass; tPearsRsePsaurlatsm.MeatexrVsi-sibleClusters;
        >RenderFlags = RasterResults.RenderFlags;


        PassParameters->ClusterPageData                         =
        Nanite::GStreamingManager.GetClusterPageDataSRV();
        PassParameters->ClusterPageHeaders =
        Nanite::GStreamingManager.GetClusterPageHeadersSRV();

        PassParameters->VisibleClustersSWHW = GraphBuilder.CreateSRV(VisibleClustersSWHW);

        PassParameters->MaterialRange =             MaterialRange;
        PassParameters->VisibleMaterials              = GraphBuilder.CreateSRV(VisibleMaterials,
        PF_R32_UINT);

        PassParameters->VisBuffer64 = VisBuffer64;// PassParametersV-isibility
        >DbgBuffer64 PassParameters->DbgBuffe=r32DbgBuffer64;
                                                = DbgBuffer32;
        PassParameters->RenderTargets = GBufferRenderTargets;//Render Texture

        // Uniform Buffer
        PassParameters->View = View.ViewUniformBuffer;// To get VTFeedbackBuffer PassParameters->BasePass =
        CreateOpaqueBasePassUniformBuffer(GraphBuilder, View, 0, {}, DBufferTextures, nullptr);


        switch(NaniteMaterialCulling) {// Usage8x4Grid
        rendering of32bit,Eachbit case1:case2:
                                                Onetile.


                PassParameters->GridSize.X          =   8;
                PassParameters->GridSize.Y break;   =   4;


        //Use64x64Pixel block rendering.
        case        3:
        case   4:

                PassParameters->GridSize = FMath::DivideAndRoundUp(View.ViewRect.Max -
```

```
View.ViewRect.Min, {64,64});
                break;

            //Use fullscreen tile rendering.
            default:
                PassParameters->GridSize.X          =  1;
                PassParameters->GridSize.Y break;   =  1;

            }

            constFExclusiveDepthStencil MaterialDepthStencil = UseComputeDepthExport()
                ?FExclusiveDepthStencil::DepthWrite_StencilNop
                : FExclusiveDepthStencil::DepthWrite_StencilWrite;

            PassParameters->RenderTargets.DepthStencil = FDepthStencilBinding(
                MaterialDepth,
                ERenderTargetLoadAction::ELoad,
                ERenderTargetLoadAction::ELoad,
                MaterialDepthStencil
            ) ;

            TShaderMapRef<FNaniteMaterialVS>NaniteVertexShader(View.ShaderMap);

            //Add renderingpass.
            GraphBuilder.AddPass(
                RDG_EVENT_NAME("Emit     GBuffer),
                PassParameters,
                ERDGPassFlags::Raster,
                [PassParameters, &Scene, NaniteVertexShader, ViewRect = View.ViewRect,
NaniteMaterialCulling](FRHICommandListImmediate& RHICmdList)
            {
                RHICmdList.SetViewport(ViewRect.Min.X, ViewRect.Min.Y,0.0f, ViewRect.Max.X,
ViewRect.Max.Y,1.0f);

                    //Manipulates global buffer parameters.

                    FNaniteUniformParameters        UniformParams;
                    UniformParams.SOAStrides        = PassParameters->SOAStrides;
                    UniformParams.MaxVisibleClusters=           PassParameters->MaxVisibleClusters;
                    UniformParams.MaxNodes =        PassParameters->MaxNodes; =
                    UniformParams.RenderFlags        PassParameters->RenderFlags;

                    UniformParams.MaterialConfig.X              =NaniteMaterialCulling;
                    UniformParams.MaterialConfig.Y          =   PassParameters->GridSize.X;
                    UniformParams.MaterialConfig.Z          =   PassParameters->GridSize.Y; 0;
                    UniformParams.MaterialConfig.W          =

                    UniformParams.RectScaleOffset = FVector4(1.0f,1.0f,0.0f,0.0f);// Render a
rect that covers the entire screen

                    //Material Clipping Mode
                    if(NaniteMaterialCulling ==3|| NaniteMaterialCulling==4) {

                        FIntPoint ScaledSize = PassParameters->GridSize *64;
                        UniformParams.RectScaleOffset.X =float(ScaledSize.X) /
float(ViewRect.Max.X - ViewRect.Min.X);
                        UniformParams.RectScaleOffset.Y =float(ScaledSize.Y) /
float(ViewRect.Max.Y - ViewRect.Min.Y);
                    }
```

```
                // ClusterPage and visibility data
                UniformParams.ClusterPageData =
                UniformParams.ClusterPageHeaders
                UniformParams.VisibleClustersSWHW
                                                    PassParameters->ClusterPageData; =
                                                        PassParameters->ClusterPageHeaders;
                                                    = PassParameters->VisibleClustersSWHW-
> GetRHI();

                //Material Data
                UniformParams.MaterialRange =       PassParameters->MaterialRange->GetRHI(); = PassParameters-
                UniformParams.VisibleMaterials        >VisibleMaterials->GetRHI();

                //VisibleUsexfaccording to
                hinumberormParams.VisBuffer64   =   PassParameters->VisBuffer64-
                > GetRHI(); UniformParams.DbgBuffer64 = PassParameters->DbgBuffer64-
                > GetRHI(); UniformParams.DbgBuffer32 = PassParameters->DbgBuffer32-
                > GetRHI(); const_cast<FScene&>

(Scene).UniformBuffers.NaniteUniformBuffer.UpdateUniformBufferImmediate(UniformParams);

                FGraphicsMinimalPipelineStateSet GraphicsMinimalPipelineStateSet;

                TArray<FNaniteMaterialPassCommand, SceneRenderingAllocator>
NaniteMaterialPassCommands;
                //BuildNaniteMaterialPasscommand.
                BuildNaniteMaterialPassCommands(RHICmdList,

Scene.NaniteDrawCommands[ENaniteMeshPass::BasePass], NaniteMaterialPassCommands);

                FMeshDrawCommandStateCache StateCache;

                constuint32 TileCount = UniformParams.MaterialConfig.Y *
UniformParams.MaterialConfig.Z;// (W * H)
                //Traverse all material channel commands and submit them one by one.
                for(autoCommandsIt = NaniteMaterialPassCommands.CreateConstIterator();
CommandsIt;     + + CommandsIt)
                {
                    SubmitNaniteMaterialPassCommand(*CommandsIt,        NaniteVertexShader,
GraphicsMinimalPipelineStateSet, TileCount, RHICmdList, StateCache);
                }
            });
        }
}
```

Before rendering BasePass, the material classification Pass needs to be executed to classify the material (Classify Material), which plays an important role in subsequent material culling operations. It uses Compute Shader to analyze the full-screen Visibility Buffer and outputs 20x12=240 pixels (called material range, format is R32G32_UINT). Each pixel (material range) encodes the material range that appears in the 64×64 area represented by each block. The colors it presents are as follows:



The Wave Operation in the above code is translated into wave operation, which is a DX concept. The corresponding concept in VK is Subgroup, which is only supported by SM6 and above. For details, please refer to the Talk of GDC2017:Wave-Programming-D3D12-Vulkan.

The source data for the drawing instructions of the Nanite material Pass in the above code is Scene.NaniteDrawCommands[ENaniteMeshPass::BasePass]. This data is generated during FPrimitiveSceneInfo::UpdateStaticMeshes. The call stack is as follows:

```
// Engine\Source\Runtime\Renderer\Private\PrimitiveSceneInfo.cpp

void FPrimitiveSceneInfo::UpdateStaticMeshes(FRHICommandListImmediate& RHICmdList, FScene* Scene,const
TArrayView<FPrimitiveSceneInfo*>& SceneInfos,boolbReAddToDrawLists) {

	(……)

	if(bReAddToDrawLists)
	{
		CacheMeshDrawCommands(RHICmdList, //cacheScene,	SceneInfos);
		NaniteDrawing instructions.
		CacheNaniteDrawCommands(RHICmdList,		Scene,	SceneInfos);
	}
}

void FPrimitiveSceneInfo::CacheNaniteDrawCommands(FRHICommandListImmediate& RHICmdList, FScene* Scene,
constTArrayView<FPrimitiveSceneInfo*>& SceneInfos) {

	(……)

	//Traverse all the scene information of the scene and build them one by oneNaniteDrawing instructions.
	for(FPrimitiveSceneInfo* PrimitiveSceneInfo : SceneInfos) {

		BuildNaniteDrawCommands(RHICmdList, Scene, PrimitiveSceneInfo);
	}

	(……)
```

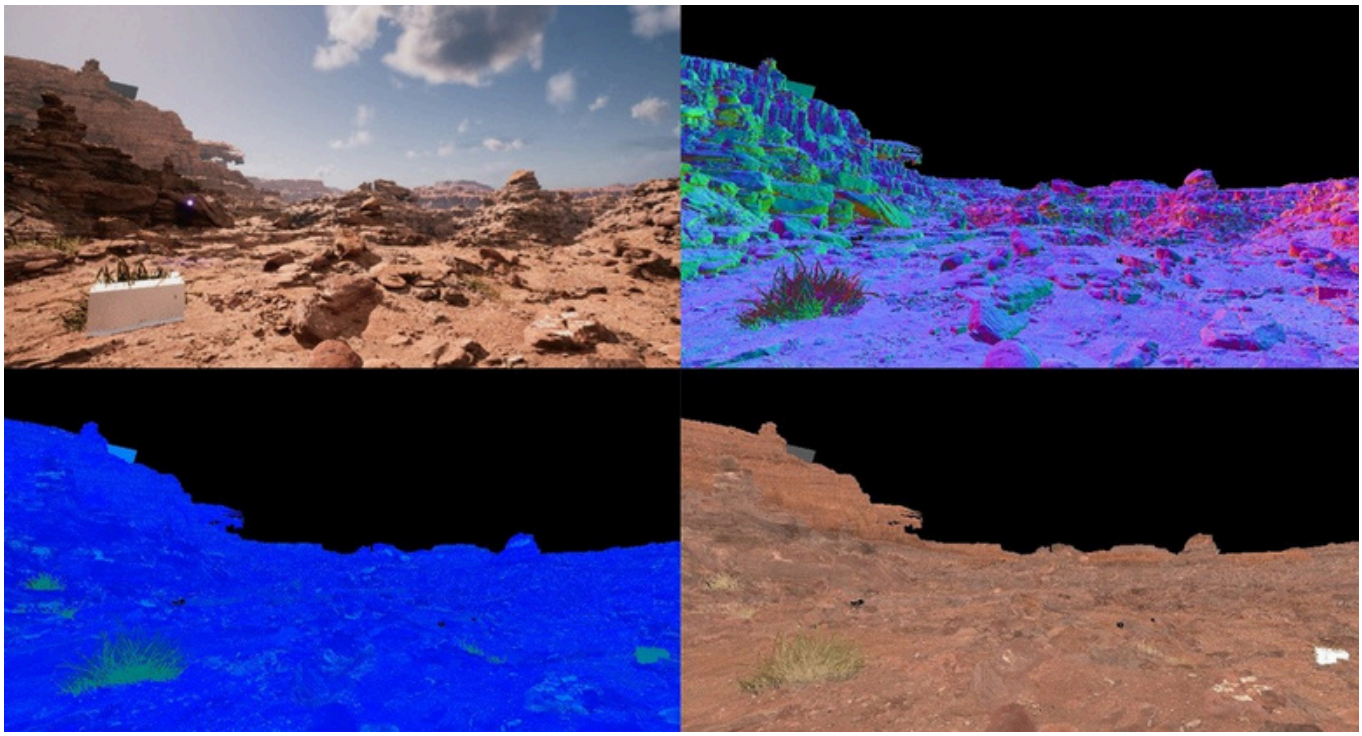
```
}

void BuildNaniteDrawCommands(FRHICommandListImmediate& RHICmdList, FScene* Scene,
FPrimitiveSceneInfo* PrimitiveSceneInfo)
{
	(……)

	for(int32 MeshPass =0; MeshPass < ENaniteMeshPass::Num; ++MeshPass) {

		FNaniteDrawListContextNaniteDrawListContext(Scene-
>NaniteDrawCommandLock[MeshPass], Scene->NaniteDrawCommands[MeshPass]);

		//createNanitePatternMeshProcessor. FMeshPassProcessor*
		NaniteMeshProcessor = nullptr; switch(MeshPass)

		{
			caseENaniteMeshPass::BasePass:
				NaniteMeshProcessor = CreateNaniteMeshProcessor(Scene, nullptr,
&NaniteDrawListContext);
				break;
			caseENaniteMeshPass::LumenCardCapture:
				NaniteMeshProcessor = CreateLumenCardNaniteMeshProcessor(Scene, nullptr,
&NaniteDrawListContext);
				break;
			default:
				check(false);
		}

		//Traverse all static grids and supportNaniteRendering grid constructionNaniteDrawing instructions.
		int32 StaticMeshesCount = PrimitiveSceneInfo->StaticMeshes.Num(); for(int32 MeshIndex =0;
		MeshIndex < StaticMeshesCount; ++MeshIndex) {

			FStaticMeshBatchRelevance& MeshRelevance = PrimitiveSceneInfo-
> StaticMeshRelevances[MeshIndex];
			FStaticMeshBatch& Mesh = PrimitiveSceneInfo->StaticMeshes[MeshIndex];

			if(MeshRelevance.bSupportsNaniteRendering) {

				uint64 BatchElementMask = ~0ull;
				//TowardsMeshProcessorAdd to grid batch, the subsequent steps are similar to traditional
				ones, no longer tracked. NaniteMeshProcessor->AddMeshBatch(Mesh, BatchElementMask,	Proxy);
				FNaniteCommandInfo CommandInfo =
NaniteDrawListContext.GetCommandInfoAndReset();
				PrimitiveSceneInfo->NaniteCommandInfos[MeshPass].Add(CommandInfo); constuint32
				MaterialDepthId = CommandInfo.GetMaterialId(); constuint32 SectionIndex =
				Mesh.SegmentIndex; PrimitiveSceneInfo->NaniteMaterialIds[MeshPass][SectionIndex] =

 MaterialDepthId;
			}
		}

		NaniteMeshProcessor->~FMeshPassProcessor();
	}

	(……)
}
```

Next, we will continue to analyze the two important interfaces of Nanite::DrawBasePass, BuildNaniteMaterialPassCommands and SubmitNaniteMaterialPassCommand:

```
// Engine\Source\Runtime\Renderer\Private\Nanite\NaniteRender.cpp

//BuildNaniteMaterialPassCommand
static void BuildNaniteMaterialPassCommands(
	FRHICommandListImmediate& RHICmdList, constFStateBucketMap&
	NaniteDrawCommands, TArray<FNaniteMaterialPassCommand,
	SceneRenderingAllocator>& OutNaniteMaterialPassCommands)

{
	OutNaniteMaterialPassCommands.Reset(NaniteDrawCommands.Num());
	FGraphicsMinimalPipelineStateSet GraphicsMinimalPipelineStateSet; constint32
	MaterialSortMode = GNaniteMaterialSortMode;

	//Traverse allNaniteDrawing instructions, construct the corresponding
	FNaniteMaterialPassCommand. for(auto& Command : NaniteDrawCommands) {

		//BuildFNaniteMaterialPassCommandExamples.
		FNaniteMaterialPassCommandPassCommand(Command.Key);


		Experimental::FHashElementId SetId = NaniteDrawCommands.FindId(Command.Key);

		int32 DrawIdx = SetId.GetIndex();
		PassCommand.MaterialDepth = FNaniteCommandInfo::GetDepthId(DrawIdx);
		//Replace the original .
		if(MaterialSortMode ==2&& GRHISupportsPipelineStateSortKey) {


			constFMeshDrawCommand& MeshDrawCommand = Command.Key;
			constFGraphicsMinimalPipelineStateInitializer&MeshPipelineState =
MeshDrawCommand.CachedPipelineId.GetPipelineState(GraphicsMinimalPipelineStateSet);
FGraphicsPipelineState* PipelineState =
PipelineStateCache::GetAndOrCreateGraphicsPipelineState(RHICmdList,
MeshPipelineState.AsGraphicsPipelineStateInitializer(),
EApplyRendertargetOption::DoNothing);
			if(PipelineState)
			{
				constuint64 StateSortKey =
PipelineStateCache::RetrieveGraphicsPipelineStateSortKey(PipelineState);
				if(StateSortKey !=0) {

					PassCommand.SortKey = StateSortKey;
				}
			}
		}

		//Add to command list.
		OutNaniteMaterialPassCommands.Emplace(PassCommand);
	}

	//Sort materials.
	if(MaterialSortMode !=0) {

		OutNaniteMaterialPassCommands.Sort();
	}
```

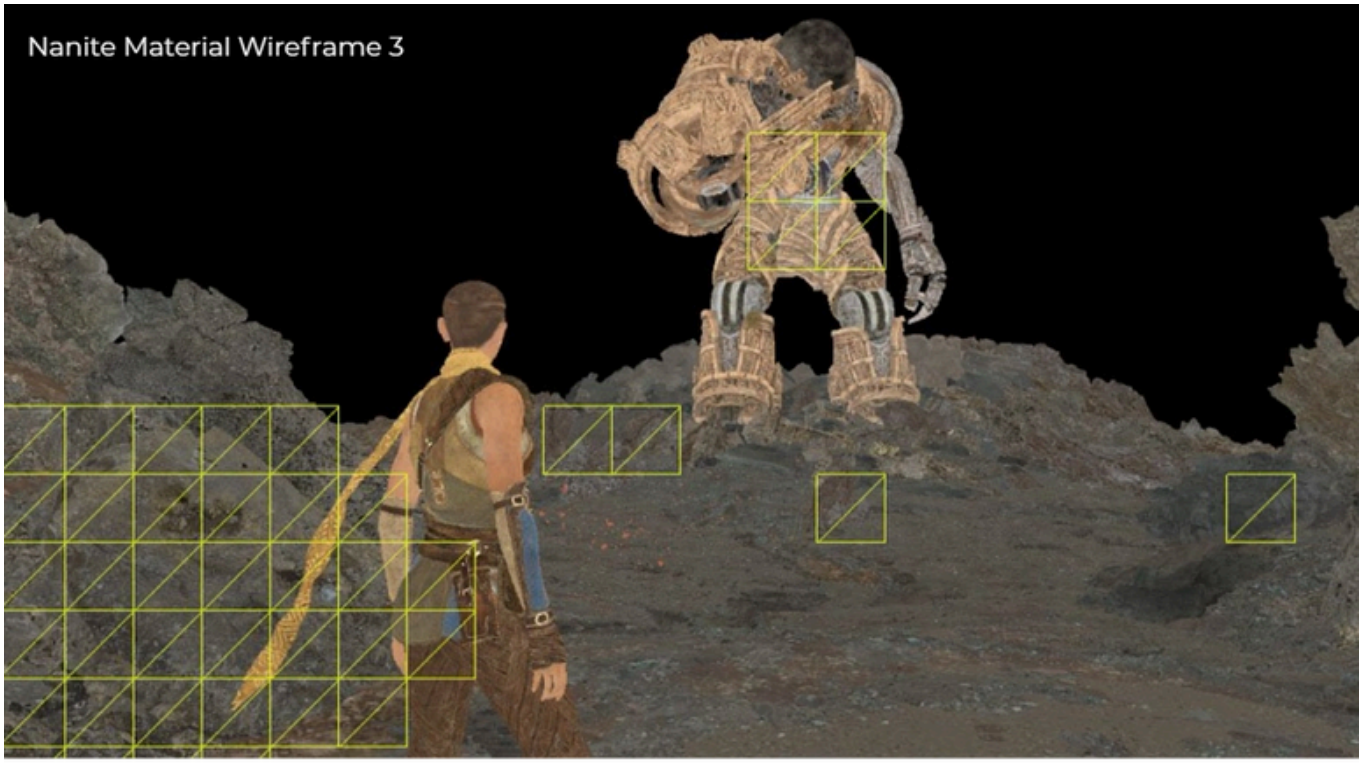
```
}

//Submit a single material channel draw command
static void SubmitNaniteMaterialPassCommand(
	constFMeshDrawCommand& MeshDrawCommand,const floatMaterialDepth,const
	TShaderRef<FNaniteMaterialVS>& FGraphicsMinimalPNipaenliinteVSetarteSxeSth&a dueinr,t32,const
	                                                GraphicsMinimalPipelineStateSet,          const
	                InstanceFactor,          FRHICommandList&          RHICmdList,
	FMeshDrawCommandStateCache& StateCache)


{
	//Submit drawing start.
	FMeshDrawCommand::SubmitDrawBegin(MeshDrawCommand,	GraphicsMinimalPipelineStateSet,
nullptr,0, InstanceFactor, RHICmdList, StateCache);

	//allNaniteThe grid drawing instructions all use the sameVS,This command has the material depth assigned at render time.
	{
		FNaniteMaterialVS::FParameters Parameters;
		Parameters.MaterialDepth = MaterialDepth;
		SetShaderParameters(RHICmdList, NaniteVertexShader,
NaniteVertexShader.GetVertexShader(), Parameters);
	}

	//Submit the drawing completed.
	FMeshDrawCommand::SubmitDrawEnd(MeshDrawCommand, InstanceFactor, RHICmdList);
}
```

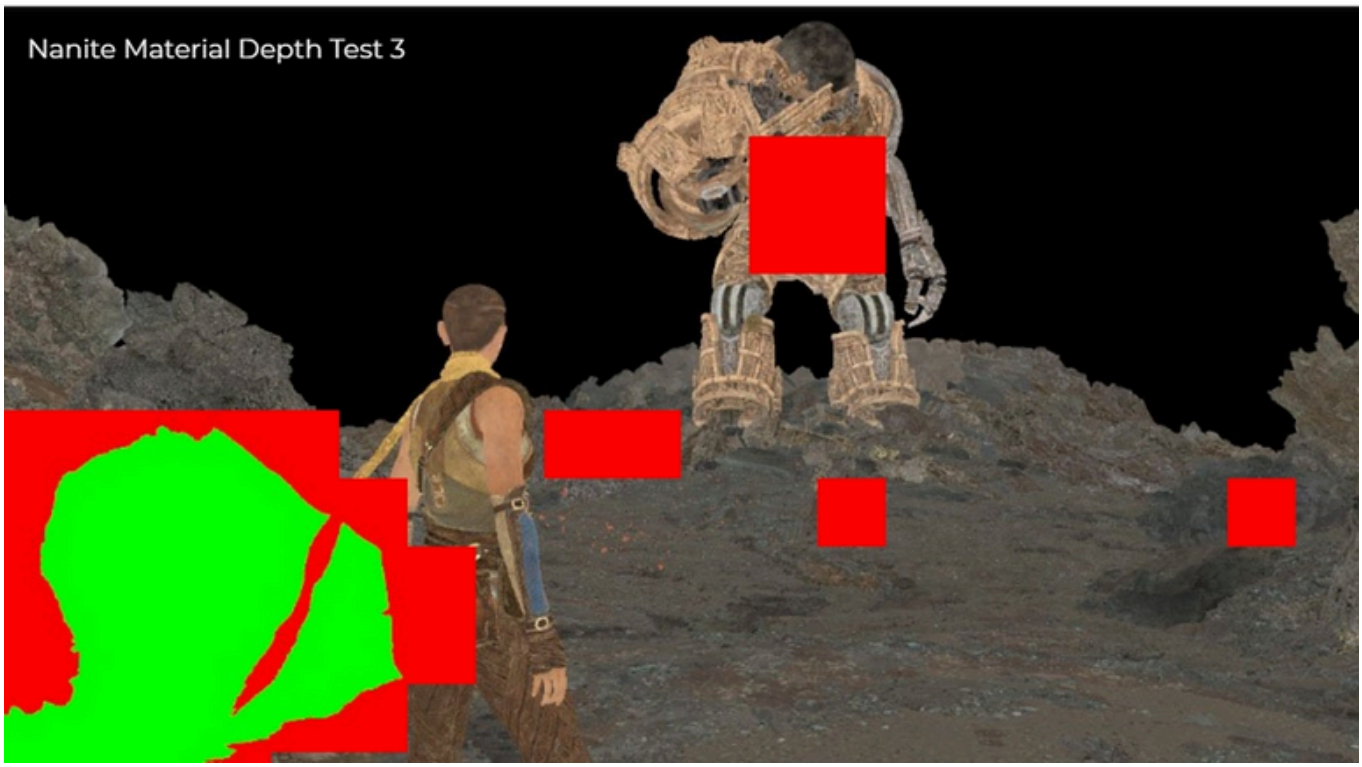
But the strange thing is that when drawing BasePass, only VS is specified, but PS is not specified. So where is PS set or is it originally empty? In order to find out the truth, RenderDoc frame analysis is used to find that PS still uses the traditional BasePassPixelShader, and the GBuffer rendered after this stage is basically the same as the traditional one:



Upper left: Rendering screen, upper right: GBufferA, lower left: GBufferB, lower right: GBufferC

Nanite submits the material as Pass during the rendering of BasePass, which means that the previously rendered material range texture and material depth can be used for fast culling, as shown in the following two pictures:





When rendering the material area of the first picture above, pixels will be quickly judged and eliminated based on the material depth and material range. As shown in the second picture, the red box indicates that all pixels covered by it have not passed the material range test and will be completely discarded by the vertex shader, while the green pixels have passed the depth test and material range test and will be sent to PS to execute the GBuffer output.

6.4.3.7 Nanite Light and Shadow

Nanite's light and shadow calculations are mixed with traditional light and shadow calculations, all in the RenderLights interface:

```
// Engine\Source\Runtime\Renderer\Private\LightRendering.cpp

void FDeferredShadingSceneRenderer::RenderLights(
	FRDGBuilder& GraphBuilder,
	FMinimalSceneTextures& SceneTextures,
	constFTranslucencyLightingVolumeTextures& TranslucencyLightingVolumeTextures, FRDGTextureRef
	LightingChannelsTexture,
	FSortedLightSetSceneInfo& SortedLightSet)
{
	(……)

	constFSimpleLightArray &SimpleLights = SortedLightSet.SimpleLights; const
	TArray<FSortedLightSceneInfo, SceneRenderingAllocator> &SortedLights =
	SortedLightSet.SortedLights;
	constint32 AttenuationLightStart = SortedLightSet.AttenuationLightStart; constint32
	SimpleLightsEnd = SortedLightSet.SimpleLightsEnd;

	(……)

	{
		RDG_EVENT_SCOPE(GraphBuilder,"DirectLighting");

		if(ViewFamily.EngineShowFlags.DirectLighting &&
			Strata::IsStrataEnabled() && Strata::IsClassificationEnabled())
		{
			//Updates the stencil buffer for all subsequentPassMark simple/complex hierarchical materials only
			once. Strata::AddStrataStencilPass(GraphBuilder, Views, SceneTextures);
		}

		(……)

		//Shadowless lighting.
		if(ViewFamily.EngineShowFlags.DirectLighting) {

			RDG_EVENT_SCOPE(GraphBuilder,"NonShadowedLights");

			(……)
		}

		//Lighting with shadows.
		{
			RDG_EVENT_SCOPE(GraphBuilder,"ShadowedLights");

			(……)

			//Draw shadows and lights with light functions.
			for(int32 LightIndex = AttenuationLightStart; LightIndex <
SortedLights.Num(); LightIndex++)
			{
				(……)

				if(bDrawShadows)
				{
					INC_DWORD_STAT(STAT_NumShadowedLights);
```

```
                (......)

                else// (OcclusionType == FOcclusionType::Shadowmap) {

                    (......)

                    //Cleaned up shadow mask texture.

                    ClearShadowMask(ScreenShadowMaskTexture);

                    //Render shadow casters.

                    RenderDeferredShadowProjections(GraphBuilder, SceneTextures,
TranslucencyLightingVolumeTextures, &LightSceneInfo, ScreenShadowMaskTexture,
ScreenShadowMaskSubPixelTexture, bInjectedTranslucentVolume);
                }

                bUsedShadowMaskTexture =true;
            }

            (......)

            if(bDirectLighting)
            {
                const boolbRenderOverlap =false;
                //Renders a single light source.
                RenderLight(GraphBuilder, SceneTextures,
                                                    &LightSceneInfo,
ScreenShadowMaskTexture, LightingChannelsTexture, bRenderOverlap);
            }

            (......)
        }
    }
}
}
```

Since the processing logic of UE5's RenderLights is highly similar to that of UE4, only the initialization of the Strata template is added. Let's continue to look at the logic of RenderLight:

```
void FDeferredShadingSceneRenderer::RenderLight(
    FRHICommandList&        RHICmdList,         const
    FViewInfo& View,constFLightSceneInfo* LightSceneInfo,
                                        FRHITexture*
    ScreenShadowMaskTexture,            FRHITexture*
    LightingChannelsTexture, bRenderOverlap,bool    bool
    bIssueDrawEvent)
{
    (......)

    //Renders the internal interface of a light source.

    autoRenderInternalLight = [&](boolbStrataFastPath) {

    (......)
    //set upStrataDepth-stencil buffer.
    if(Strata::IsStrataEnabled() && Strata::IsClassificationEnabled()) {
```

```
            GraphicsPSOInit.DepthStencilState = TStaticDepthStencilState<
            false, CF_Always,true, CF_Equal, SO_Keep, SO_Keep,
            SO_Keep,true, CF_Equal, SO_Keep, SO_Keep, SO_Keep,
            Strata::StencilBit, 0x0>::GetRHI();

        }
        else
        {
            GraphicsPSOInit.DepthStencilState                = TStaticDepthStencilState<false,
            CF_Always>::GetRHI();
        }

        (......)

        if(LightProxy->GetLightType() == LightType_Directional) {

            (......)

            else
            {
                (......)

                FDeferredLightPS::FPermutationDomain PermutationVector;

                (......)

                //AddedStrataTier) sort
                PermutationVector.Set<          FDeferredLightPS::FStrata >(Strata::IsStrataEnabled());
                PermutationVector.Set<          FDeferredLightPS::FStrataFastPath >
(Strata::IsStrataEnabled() && Strata::IsClassificationEnabled() && bStrataFastPath);

                TShaderMapRef< FDeferredLightPS >PixelShader( View.ShaderMap,
PermutationVector );

                (......)
            }

            (......)

            //set upStrataTarget value.
            RHICmdList.SetStencilRef(bStrataFastPath ? Strata::StencilBit :0u);

            //Draw a parallel light full screen.
            DrawRectangle(
                RHICmdList,
                0,0,
                View.ViewRect.Width(),          View.ViewRect.Height(),
                View.ViewRect.Min.X, View.ViewRect.Min.Y,
                View.ViewRect.Width(), View.ViewRect.Height(),
                View.ViewRect.Size(),
                GetSceneTextureExtent(),
                VertexShader,
                EDRF_UseTriangleOptimization);
        }
        else                //          Non-parallel light (local light source)
        {
            (......)
```

```
            TShaderMapRef<TDeferredLightVS<true> > VertexShader(View.ShaderMap);

            //Whether the camera is inside the light source geometry.
            const boolbCameraInsideLightGeometry = ((FVector)View.ViewMatrices.GetViewOrigin() -
LightBounds.Center).SizeSquared() < FMath::Square(LightBounds.W *1.05f+ View.NearClippingDistance *2.0f)

                || !View.IsPerspectiveProjection();

            //Sets the bound geometry rasterization and depth state, wherebCameraInsideLightGeometryEnter here.
            SetBoundingGeometryRasterizerAndDepthState(GraphicsPSOInit, View,
            bCameraInsideLightGeometry);

            (......)
            else
            {
                (......)

                //                  Strata.
                PermutationVector.Set<      FDeferredLightPS::FStrata >(Strata::IsStrataEnabled());
                PermutationVector.Set<      FDeferredLightPS::FStrataFastPath >
(Strata::IsStrataEnabled() && Strata::IsClassificationEnabled() && bStrataFastPath);

                TShaderMapRef< FDeferredLightPS >PixelShader( View.ShaderMap,
PermutationVector );

                (......)
            }

            (......)

            RHICmdList.SetStencilRef(bStrataFastPath ? Strata::StencilBit :0u);

            (......)

            //Choose different shapes to draw according to different types of local light.
            if(LightProxy->GetLightType() == LightType_Point ||
                LightProxy->GetLightType() == LightType_Rect )
            {
                StencilingGeometry::DrawSphere(RHICmdList);
            }
            else            if          ( L ight Proxy -> G et L ight Type ( )                                    = =       L
            {
                StencilingGeometry::DrawCone(RHICmdList);
            }
        }
        }   ;

        //Call onceStrataVersion of the light source drawing (UE4light source calculation mode).
        RenderInternalLight(false);

        //If it is turned onStrata,Then call it againStrataVersion of the light source drawing.
        if(Strata::IsStrataEnabled() && Strata::IsClassificationEnabled()) {

            RenderInternalLight(true);
        }
}
```

The lighting Shader code only adds support for Strata, which will not be discussed here.

In addition, it is worth mentioning that UE5's shadow calculation uses**Virtual Shadow Map (VSM)** technology, which is a new shadow projection method for providing consistent, high-resolution shadows, dynamic lighting with movie-quality assets and large open worlds.

VSM was first proposed by Markus Giegl et al. in 2007, and they published the paperQueried Virtual Shadow Maps, and then published the improved paperFitted Virtual Shadow Maps. Many years later, in 2015, Olsson Ola et al. combined rendering technologies such as Clusterred and published the paper More efficient virtual shadow maps for many lights.
The core of this technology is that it renders shadow maps in an adaptive way, that is, creating larger shadow map resolutions where needed, without storing information from the previous frame, making it suitable for fully dynamic scenes. Therefore, it can guarantee sub- Pixel accuracy queries of shadow maps, eliminating projection and perspective aliasing of traditional shadow maps.
VSM uses**Virtual Tiled Shadow Mapping**technology. The algorithm is described as follows:

- Allocate the maximum texture resolution that the GPU can support (generally taken in the early days4096×4096,now16*k*×16*k*or higher).
- Divide the shadow map along the X and Y directions into*n*×*n*(like16×16) are divided into tiles of equal size (each tile uses texels of the maximum shadow map texture resolution), so the effective resolution of the maximum shadow map is equivalent to (16×4096) × (16×4096) =65536×65536. For each block:
  - Render shadows to a shadow map texture (clipped using the tile's light frustum, and overwriting the previous tile's shadow map).
  - Use it immediately to mask the parts of the scene that are covered by the current shadow map.

(a) 4096^2 conventional SM

(b) 32×32 2048^2 QVSM

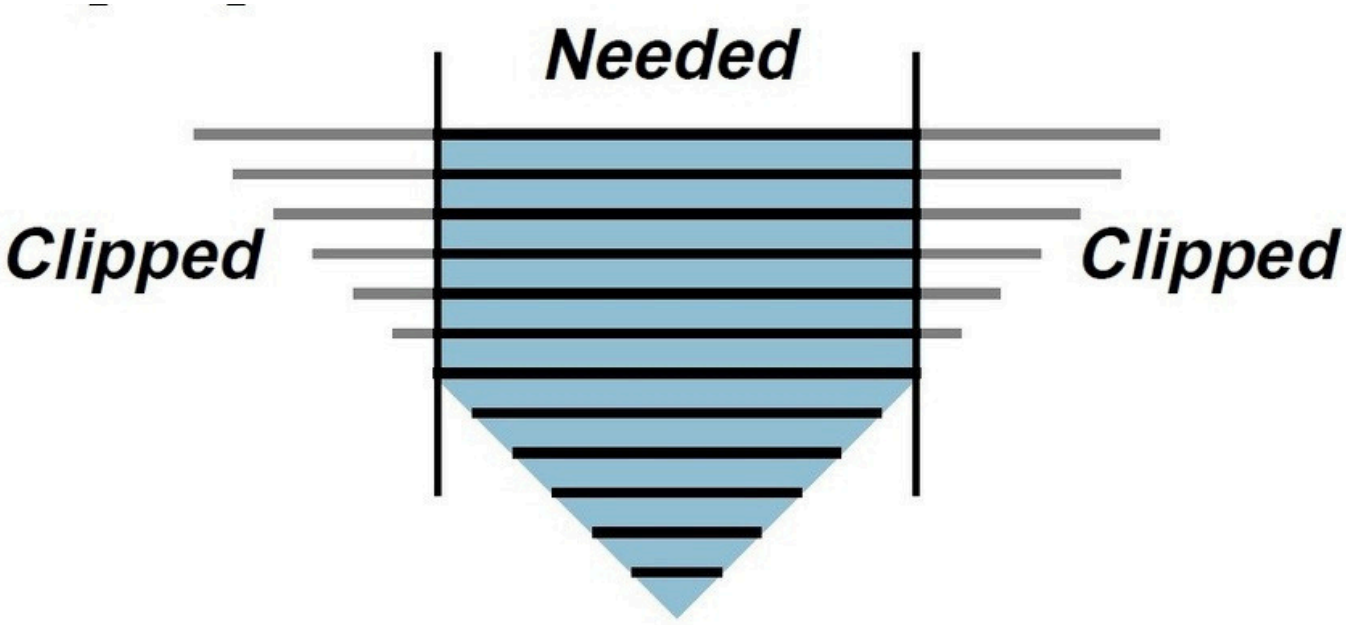
Top: Using traditional shadow maps, serious aliasing problems occur; Bottom: Using 32x32 2048x2048 QVSM, shadow accuracy is greatly improved.

In the UE5 implementation, the maximum resolution of VSM is16*k*×16*k*pixels, each block (page) size is128×128, in order to keep performance high at a reasonable memory cost. Tiles are allocated

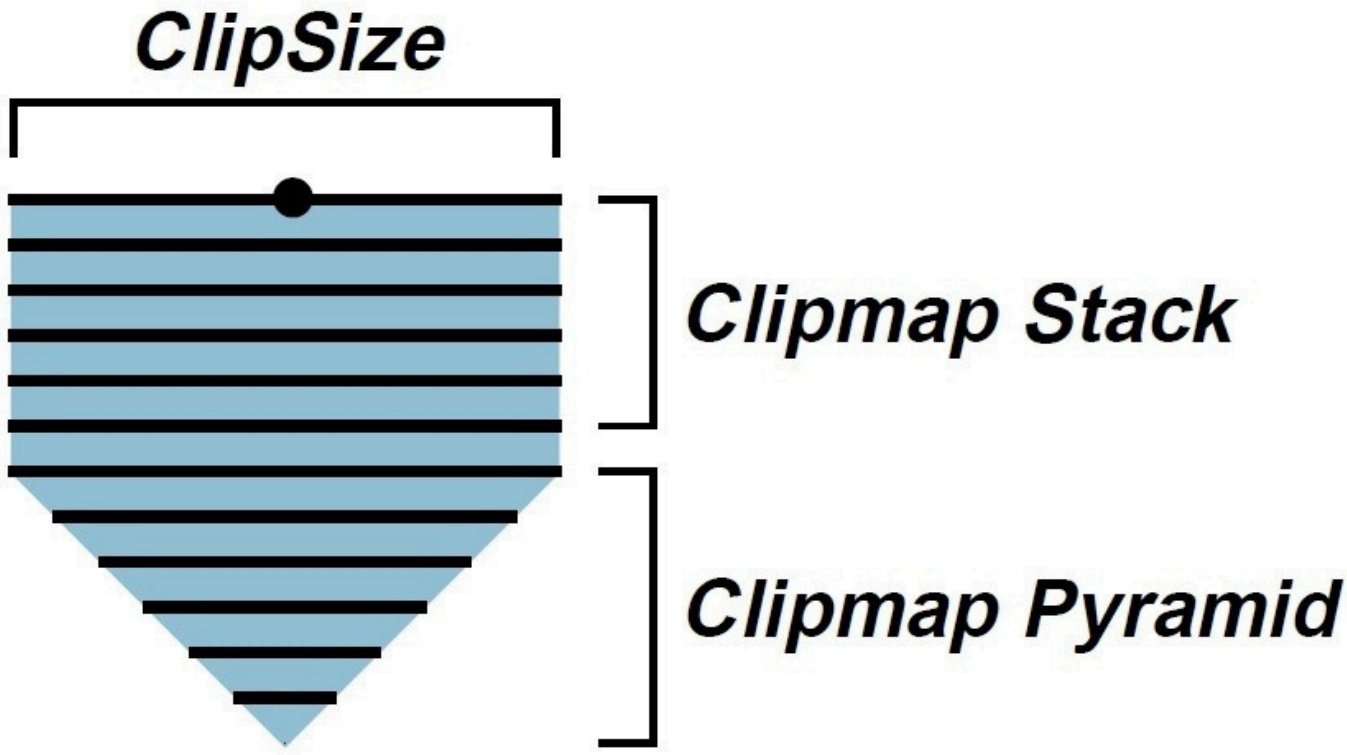
and rendered only based on the pixels on the screen that need to be shaded (based on analysis of the depth buffer). Tiles are cached between frames unless the objects or lights they refer to move, which further improves performance.

In addition, UE5 adopts ClipMap technology for directional light shadows to replace CSM to obtain higher shadow map resolution. ClipMap was first proposed by Christopher C. Tanner et al. in the

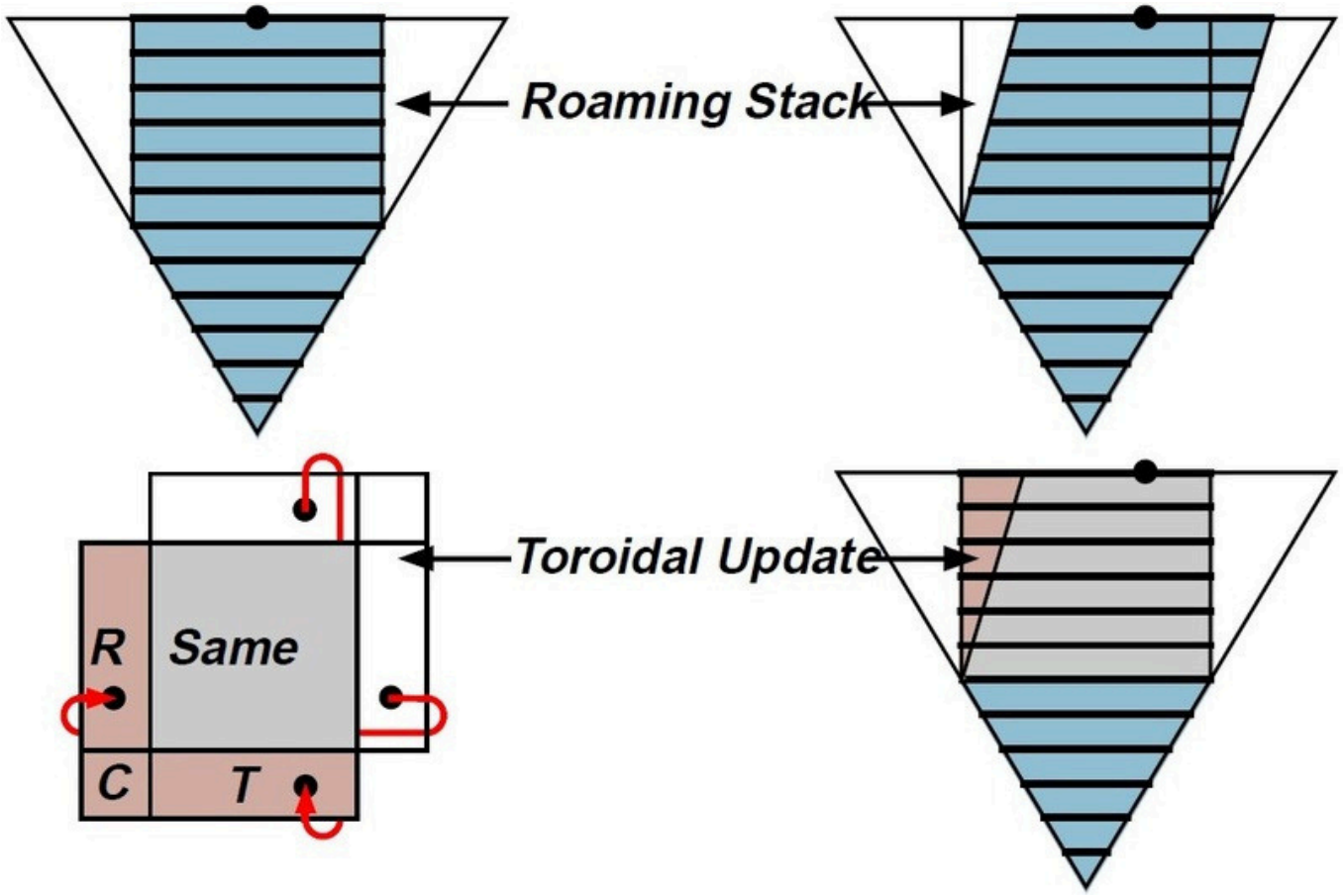
paperThe clipmap: a virtual mipmapin 1998. The core of this technology is to set an upper limit on the size of the shadow map mipmap. Mipmaps exceeding this upper limit will be clipped (not loaded into memory):



This forms the Clipmap Stack and Clipmap Pyramid:



When the camera (field of view) changes, you need to modify the area of the remapped Clipmap Stack and load the remapped Clipmap data so that the Clipmap Stack part corresponds to the field of view:



Schematic diagram of Clipmap update after the field of view changes. Toroidal Update is used here to improve performance.

6.4.4 Nanite Summary

Nanite technology involves pre-processing construction before rendering, granularity clipping at all levels during rendering, rasterization, BasePass and Lighting stages. During this period, a large number of data structures, algorithms, rendering technologies and corresponding optimization technologies were applied.

Nanite does not use Geometry Image technology as previously reported, but instead uses rough representations of various levels based on Cluster, ClusterGroup, and Page. This technology can make full use of pre-calculation to build simplified data and corresponding storage data in advance, so as to more efficiently reconstruct, index, process and render Nanite data during rendering, but it also leads to the disadvantage that Nanite only supports static meshes.

Nanite's rendering stage is interspersed in the traditional rendering pipeline, and goes through GPUScene update, stream management, clipping, rasterization, BasePass and Readback stages successively, giving full play to the power ofGPU-Driven Rendering Pipeline, and finally presenting Nanite's data well on RenderTrage. Each step has gone through numerous passes, rendering technologies and optimization techniques. For ~~example, there are different granu~~larity clipping such as instance-by-instance, cluster-by-cluster, page-by- page and triangle-by-triangle, all of which are GPU-driven clipping to reduce CPU and GPU IO. The rasterization stage uses a mixed relationship of CS soft rasterization + PS hard rasterization by default, in which CS soft rasterization is responsible

for the rasterization of triangles with very small areas (to avoid Quad Overdraw), and PS is responsible for the rasterization of triangles with larger areas. After rasterization, only the triangle ID and depth are output (Visibility Buffer technology) to reduce GBuffer occupancy and bandwidth consumption. The output of BasePass is the same as the traditional one, stored in GBufferA, GBufferB... In the subsequent lighting calculation stage, except for the support of Strata mode, other lighting logic is basically the same as the traditional one.

In order to improve the quality of shadows and optimize shadow consumption, UE5 uses VSM and Clipmap technology to achieve real-time shadow rendering with a balanced effect and consumption.

The following chapters will be presented in UE5 Special Part 2:

6.5 Lumen

6.6 Other Rendering Techniques

6.7 Summary

-
-
-
-
-

References

- Unreal Engine Source
- Rendering and Graphics
- Materials
- Graphics Programming
- New Rendering Features

- Nanite Virtualized Geometry
- Lumen Global Illumination and Reflections
- Lumen Technical Details
- Behind the scenes of “Lumen in the Land of Nanite” | Unreal Engine 5
- Unreal Engine 5 Early Access Release Notes
- Virtual Shadow Maps
- A First Look at Unreal Engine 5
- Unreal Engine 5 download learning resources summary
- How do you rate Unreal Engine 5 Early-Access?
- Unreal Engine 5 How to Nanite Skeletal Mesh A Brief
- Analysis of UE5 Nanite Implementation Optimization
- technology behind UE5 Nanite and Lumen
- Game Engine Essay 0x20: UE5 Nanite Source Code Analysis Rendering: BVH and Cluster Culling
- Learning Boundary Edges for 3D-Mesh Segmentation
- Family of Graph and Hypergraph Partitioning Software METIS
- Three Phases Coarsening Partitioning Uncoarsening
- METIS: Unstructured Graph Partitioning and Sparse Matrix Ordering System
- Dynamic LZW for Compressing Large Files
- Geometry images
- New Quadric Metric for Simplifying Meshes with Appearance Attributes
- Lock Mesh Edges
- Wave-Programming-D3D12-Vulkan
- GPU-Driven Rendering Pipeline
- Optimizing the Graphics Pipeline with Compute A
- Macro View of Nanite
- DynamicOcclusionWithSignedDistanceFields
- Queried Virtual Shadow Maps
- Fitted Virtual Shadow Maps
- More efficient virtual shadow maps for many lights
- The clipmap: a virtual mipmap
-

https://github.com/pe7yu