Unreal Engine 5.4 Raytracing Guide

July 1, 2024

Introduction

UE5 ships with new rendering technology, mainly **Lumen** for lighting and **Nanite** for scalable geometry rasterizing. With those changes, raytracing settings and workflow are changed to support the new tech. This document tries to cover the important details about hardware raytracing in UE5. This document should not be considered as a replacement of the Epic's official documentation. This document is written to complement the information covered official documentation about the topic of hardware raytracing. So, it is recommended to refer to the official documentation links referenced through this document.

Hardware Ray Tracing and Path Tracing Features | Epic Developer Community (epicgames.com)

This document refers to some advanced features and enhancements available only in the **Nvidia RTX branch of UE5 (NVRTX)**. To learn more details about these features, it is also recommended to refer to the NVRTX documents to learn more about its features.

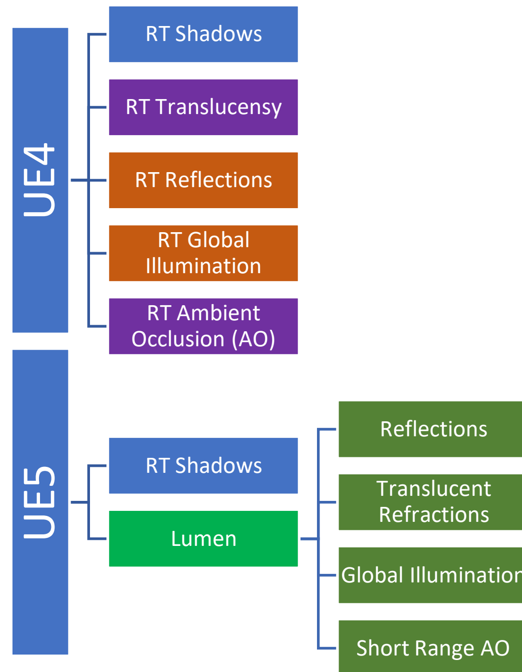
NvRTX/UnrealEngine: Unreal Engine source code (github.com)

Raytracing in UE5 vs UE4

One of the main differences between UE4 and UE5 is the introduction of Lumen. Lumen combines two key rendering components (reflections and global illumination) in one system instead of two separate systems as in UE4. This allows re-use of data structures for efficiently solving multiple rendering components.

Lumen is heavily dependent on raytracing. For hardware that doesn't support raytracing, Lumen utilizes software raytracing (SW RT) which requires mesh Distance Fields (DF) to be generated for all meshes in the scene. However, when Hardware Raytracing (HW RT) is available and enabled, Lumen leverages it to improve accuracy and visual fidelity.

The following graph summarizes how raytracing effects are managed in UE5 vs UE4.



RT effects highlighted in orange are deprecated in UE5 and dropped in UE5.4 as they are replaced by Lumen sub-systems. RT Translucency is still available in addition to RT Ambient Occusion which requires Lumen GI to be disabled.

Enabling Raytracing in UE5

The following steps are a quick guide for enabling hardware raytracing in UE5:

1. Hardware Raytracing

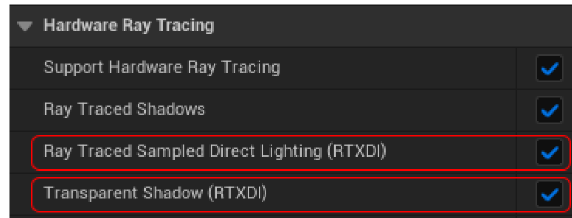
The first step is to make sure **Support Hardware Ray Tracing** is enabled in the project settings under the **Rendering** section. This requires few other options to be enabled including the **Support Compute Skinned Cache** and having the **Default RHI** set be set to **DirectX 12**. Once these options are set, UE5 rendering systems can leverage hardware raytracing.

2. Shadows

To enable raytraced shadows in UE5, simply enable the **Ray Tracing Shadows** option in the project settings under the **Rendering** section. Advanced raytraced shadow options for controlling quality and performance will be covered in a later section.

The NVRTX branch has an additional option for **Ray Traced Sampled Direct Lighting (RTXDI)**. This feature allows rendering high numbers of accurate area light shadows in real-time at relatively fixed performance cost with visuals comparable to offline path-tracing.

Additionally, RTXDI supports rendering dynamic and accurate colored shadows when light passes through transparent material. This feature can be enabled in the project settings by checking the **Transparent Shadow (RTXDI)** option.



The following links provides more information about Sample Lighting and its integration in NVRTX:

- NVIDIA RTXDI | NVIDIA Developer
- UnrealEngine/Docs/RTXDI at nvrtx-5.3 · NvRTX/UnrealEngine (github.com)

3. Global Illumination (GI)

Lumen is the default system in UE5 for dynamic GI. In the project settings, make sure that the **Use Hardware Ray Tracing** when available is enabled under the Lumen sub-section.

In NVRTX, Lumen GI benefits from several optimizations and enhancements including:

- Shader Execution Reordering (SER) for reducing the cost of sorting when evaluating shaders. In certain cases, this feature can lead to significant performance boost specifically with Lumen Reflections
- Leveraging **ReSTIR** to accelerate Lumen GI allowing it to scale with the high light count permitted by RTXDI

4. Reflections

Similar to GI, Lumen is set as the default system for rendering reflections, so enabling the **Use Hardware Ray Tracing when available** option allows reflections to be rendered using HW RT.

To get accurate HW RT reflections, it is recommended to switch the **Ray Lighting Mode** to **Hit Lighting for Reflections**.

Similarly, it is possible to get better reflections on translucent materials by enabling **High Quality Translucency Reflections**.

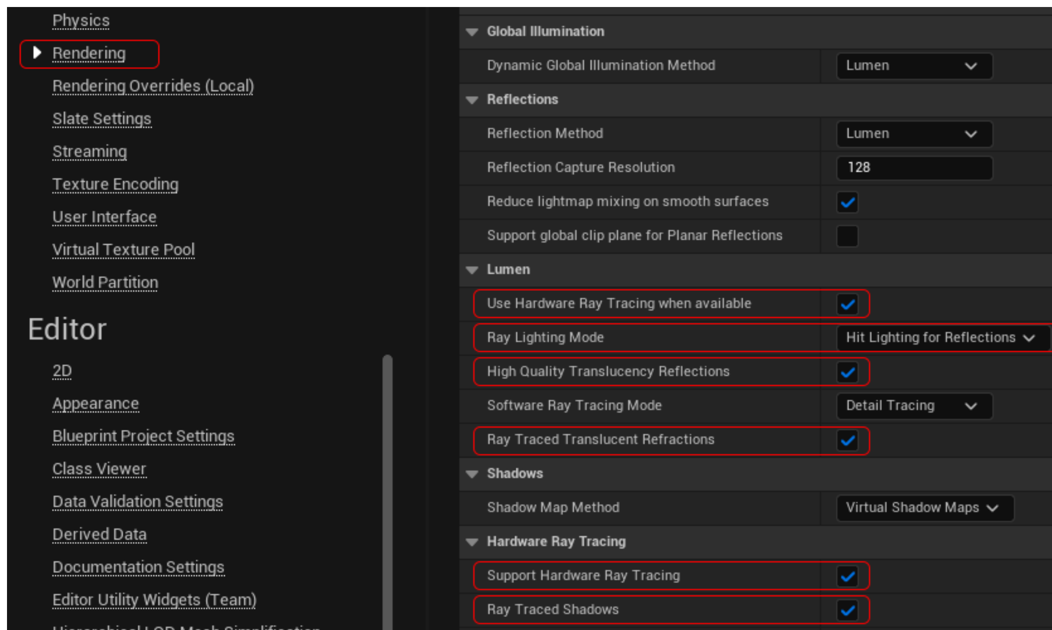
Keep in mind that the previous options (i.e. Hit Lighting and High Quality Translucency Reflections) have higher GPU cost.

With NVRTX, Lumen reflections benefit from the same optimizations listed previously for Lumen GI.

5. Refractions

Enable **Ray Traced Translucent Refractions** to leverage HW RT in Lumen when rendering translucent materials. This requires **Hit Lighting** to be enabled and increases the GPU cost.

The following screenshot highlights all the previous options that are required to enable raytracing in UE5.



Lumen Raytracing Overview

This section goes into more detail about the Lumen pipeline and its data structure. The goal is to provide high-level understanding about how it works and how hardware raytracing is essential to get the best quality and performance out of Lumen.

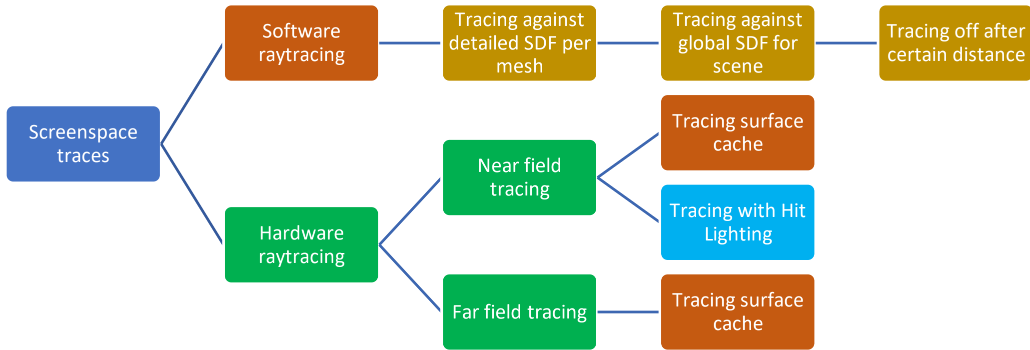
As mentioned earlier Lumen solves reflections and GI differently from the UE4 raytracing counterparts. Lumen generates a data structure from scenes' surfaces called the **Surface Cache** to perform expensive raytracing and shading calculations in a more efficient way. When the Surface Cache is visualized, it resembles a pre-rendered low-res version of the actual scene which is referred to as the **Lumen Scene**. The Lumen scene is automatically generated; however, users have few options to control/debug and fix the generated scene if necessary, as will be covered in later sections.

For reflections, by default Lumen utilizes the Surface Cache for both hardware and software raytracing. Generally, the reflection quality from the surface cache is not accurate due to the simplified nature of the Lumen Scene. For most cases, Surface Cache might work fine for blurry off-screen reflections, however for sharper reflections (e.g., mirrors) it may not be sufficient.

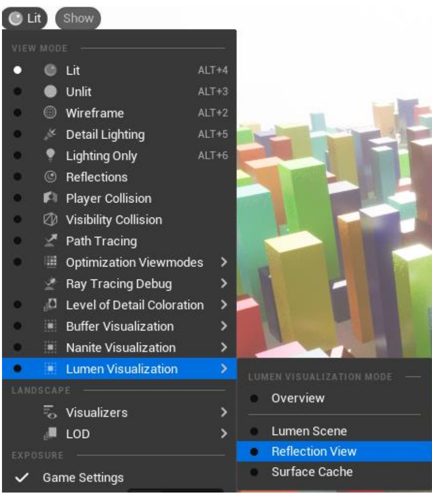
To address this issue, Lumen does screen space traces using rendered GBuffers to provide better quality reflections when possible while keeping Surface Cache for off screen only.

In the case of using HW RT, Lumen provides high quality reflection mode called **Hit Lighting** where the actual material is evaluated instead of the Surface Cache. This mode provides the best and most accurate reflections, at the cost of additional GPU performance.

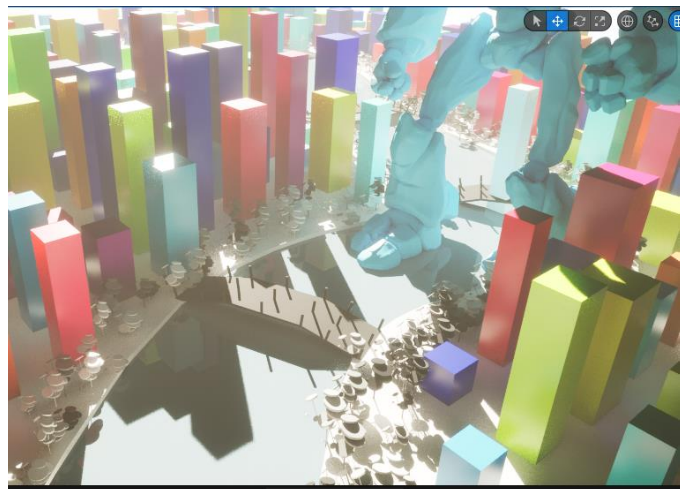
The following diagram explains in high-level how Lumen works and where it uses hardware raytracing:



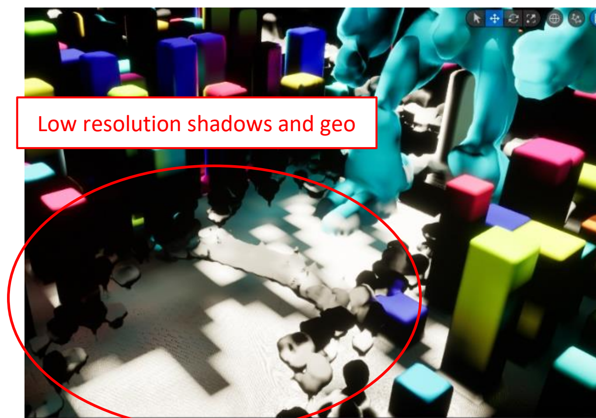
It is worth noting that HW RT generally provides better quality Surface Cache than SW RT. It is possible to visualize the difference by using the Lumen reflections view:



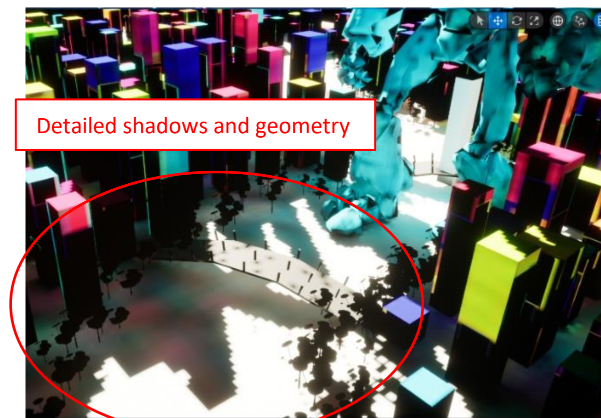
Lumen Visualization Options



Lit Scene

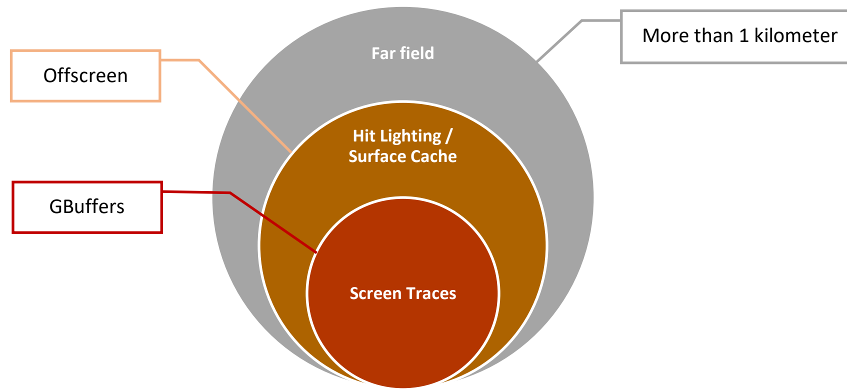


Visualization for Lumen Software RT



Visualization for Lumen Hardware RT

To keep Lumen scalable in large open-world environments using the World Partition, after the ray tracing scene radius -which is by default 1km from the camera-, rays will trace against another simplified version of the scene referred as the **Far Field** to extend global illumination and reflections at a cheaper cost. The Far Field makes use of Hierarchical Level of Detail (HLOD) meshes generated by World Partition. The HLOD1 mesh is used for Far Field representation.



For additional details about Lumen and its sub-systems it is recommended to refer to the following documents:

- Lumen Technical Details | Epic Developer Community (epicgames.com)
- Lumen Performance Guide | Epic Developer Community (epicgames.com)

Nanite Raytracing Overview

Nanite is a new approach for efficiently storing and rasterizing dense meshes where theoretically a triangle could be the size of a pixel. Nanite meshes are supported in raytracing effects and are enabled by default.

Fallback Mesh

To keep raytracing performance scalable, Nanite generates a decimated version of the original mesh referenced as **Fallback Mesh** that is used with HW raytracing effects.

For low-frequency effects such as GI, rough reflections or smooth area shadows, any disparity between the rasterized Nanite mesh and the ray traced fallback mesh unlikely to be noticeable; however, this might be an issue with RT effects that require accuracy such as hard shadows and mirrored reflections.

For hard shadows where self-shadowing might be an issue as shown in the screenshots below, it is possible to minimize the self-intersection by increasing the trace distance using the following CVar:

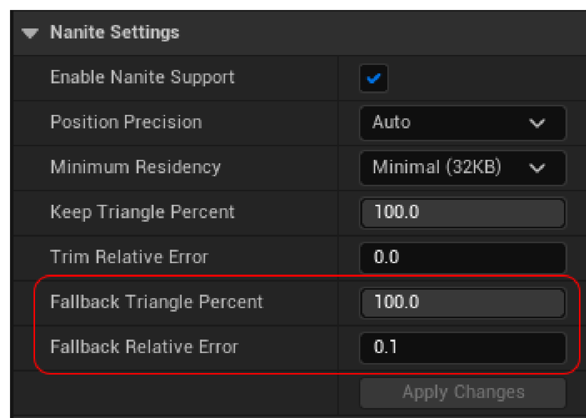
```
r.RayTracing.Shadows.AvoidSelfIntersectionTraceDistance
```

Self-shadowing issue due to silhouette differences between Nanite and fallback mesh

Solving self-shadowing by tweaking the Trace Distance CVar

For mirrored reflections where the visible mismatch is a noticeable issue, it is possible to control the decimation to have improve the fidelity of the fallback mesh. This can be done either by:

- Modifying the ‘Fallback Relative Error’. Smaller error values increase the triangle count of the fallback mesh.
- Modifying the ‘Fallback Triangle Percent’. Higher percentage values increase the triangle count.



Keep in mind that the triangle count of the fallback mesh directly affects the raytracing performance. So, it is recommended to increase it carefully only for hero objects.

Streamed Out Mesh

For cases where visual fidelity is a greater priority than performance, Nanite supports streaming out high LoD cut of the mesh to be used for raytracing effects. This can be done by setting the following Console Variable (CVar): `r.RayTracing.Nanite.Mode 1`

Keep in mind that the Nanite streamed mode will generally perform slower than the Nanite Fallback mesh due to the higher cost of building raytracing acceleration structure for denser geometry. Furthermore,

enabling this mode may substantially increase the memory footprint required by the denser geometry it utilizes.

For more details and options about Nanite it is recommended to refer to the UE5 documentation:

- Nanite Virtualized Geometry | Unreal Engine Documentation
- Nanite Virtualized Geometry | Epic Developer Community (epicgames.com)

Basic Raytracing Settings

For full control on raytracing visuals and performance, raytracing effects can be controlled on a granular level (e.g., per actor, component, light, … etc) or spatially using localized post-process volumes. In general, this is still similar to UE4 but re-directed to be used with the new UE5 tech (i.e. Lumen).

Raytracing Culling

Ray tracing requires objects outside of the camera view to be present in the scene, especially for highly reflective surfaces. This increases the cost of rendering the scene. Special culling modes are available to help with optimizing performance with minimal visuals impact.

Ray tracing culling is enabled by default and it is possible to control the culling mode and settings through the CVars: `r.RayTracing.Culling.*`

For detailed information about the culling modes and their settings, it is recommended to refer to the following document:

https://dev.epicgames.com/documentation/en-us/unreal-engine/ray-tracing-performance-guide-in-unreal-engine#culling

Although culling is essential for keeping the RT performance under control. It has overhead that can float in scenes with excessive instance count. For such scenes with many disparate parts, it is possible to create ray tracing groups to cull them as an aggregate. Actors need to be assigned the same group ID (other than -1) to be culled as a single object. This help minimizing the culling overhead and allow prioritize culling using Raytracing Group Culling Priority.

Raytracing Visibility

By default, raytracing is enabled for all meshes so they are included in the raytracing acceleration structure which allows them to be visible in any raytracing effect.

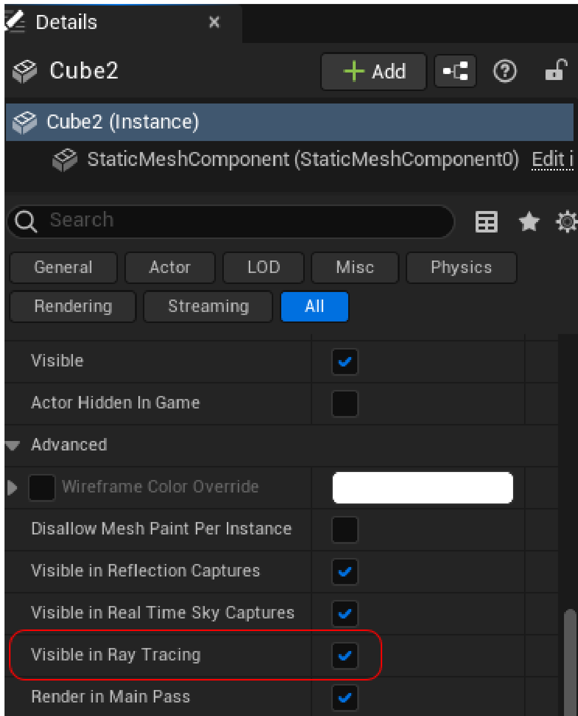
Raytracing can be toggled by geometry type through CVars: `r.RayTracing.Geometry.*`

For static meshes, it is possible to override RT at multiple levels as listed below:

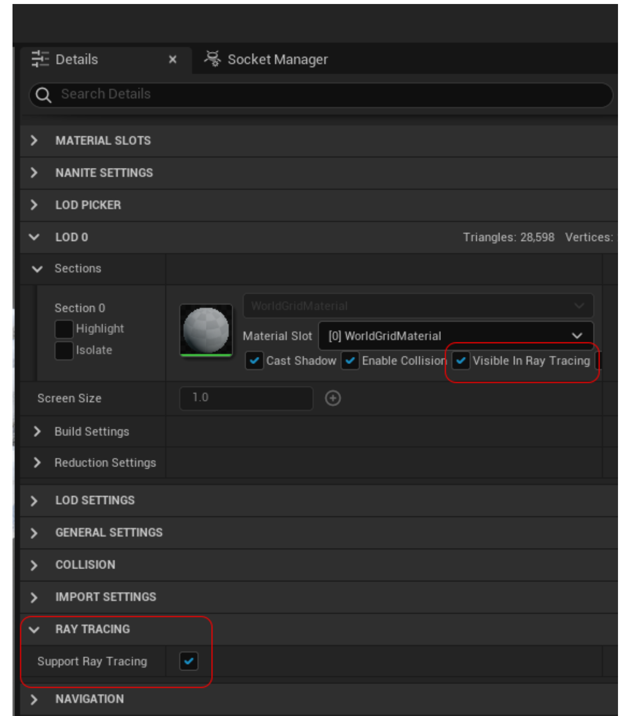
Type	Level
Actors and Components	A specific actor in a level or component in an actor class
Mesh Assets	All actors/components referencing the asset
Material Sections	All actors/components referencing the asset

Worth mentioning that controlling raytracing visibility just on a material section is generally less efficient than disabling it at the mesh object or actor level. For example, if an object has two materials where one

is invisible in raytracing, extra raytracing cost accrued due to the special way that segment of the mesh is made invisible.

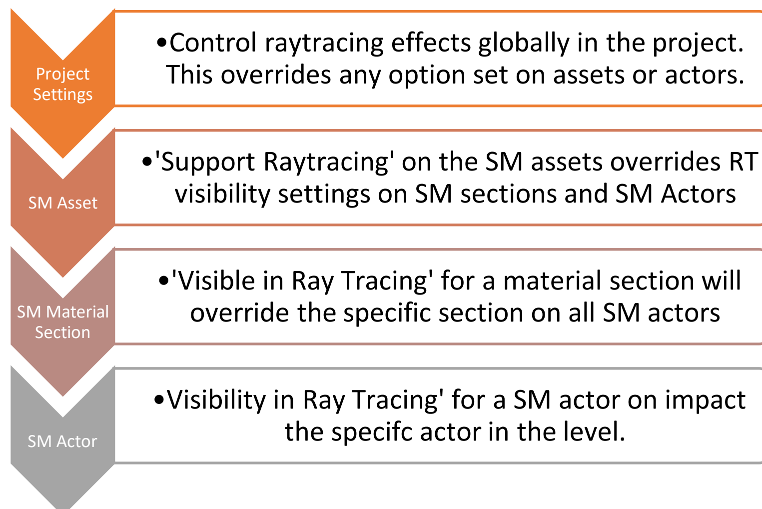


Raytracing visibility on level actors



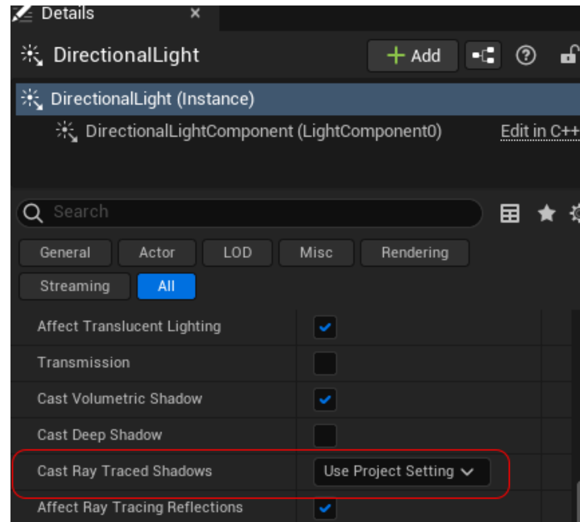
Raytracing visibility on static mesh sections

The following graph illustrates the order of raytracing visibility for Static Meshes (SM).

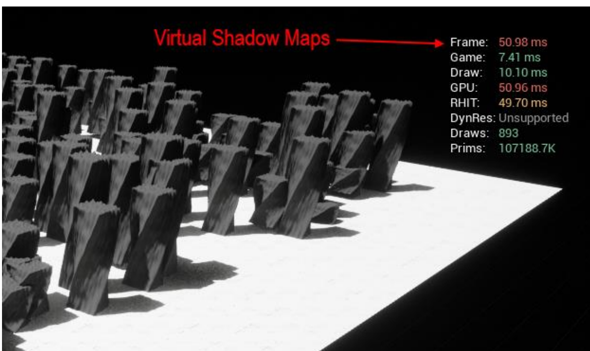


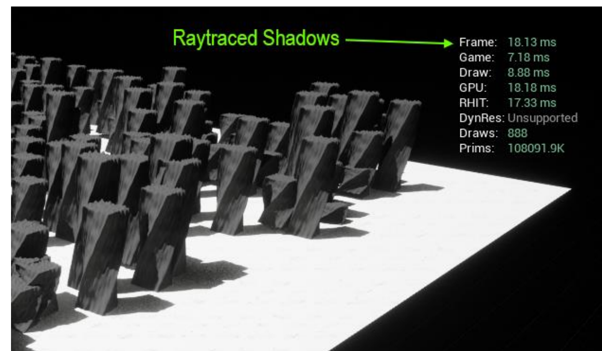
Shadows Overrides

By default, all lights inherit the raytracing shadows setting from the project settings. It is possible to override this setting for specific light actors in the level. However, disabling raytraced shadows on a light will make it fallback to (virtual) shadow maps.

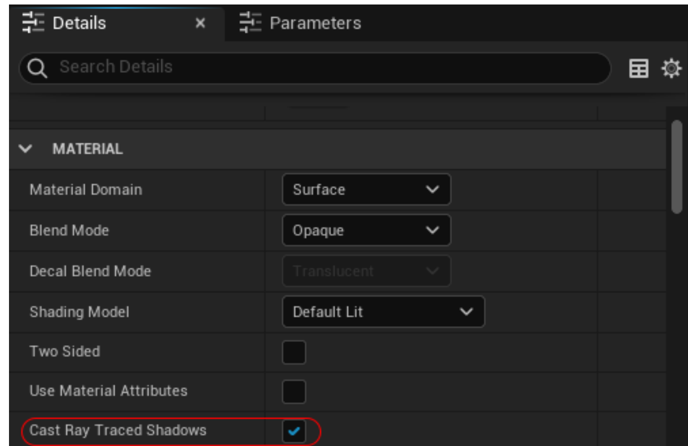


Keep in mind that in some cases raytraced shadows might be more performant than virtual shadow maps. The following example shows a case with many dense static mesh actors and one animated directional light. Raytraced shadows performs much than virtual shadow maps shadows as shown below.

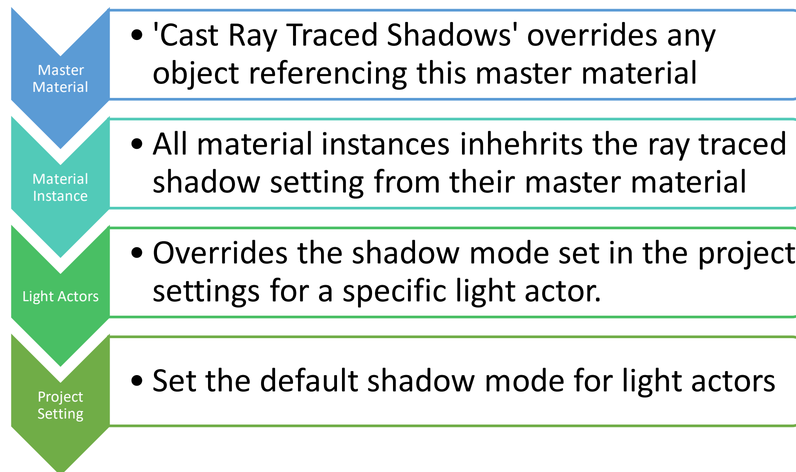




On materials, it is possible to toggle raytraced shadows for a certain material. This will impact all meshes referencing the material and its instances. Take note that it is not possible to override this option on material instances. This only works on master materials.

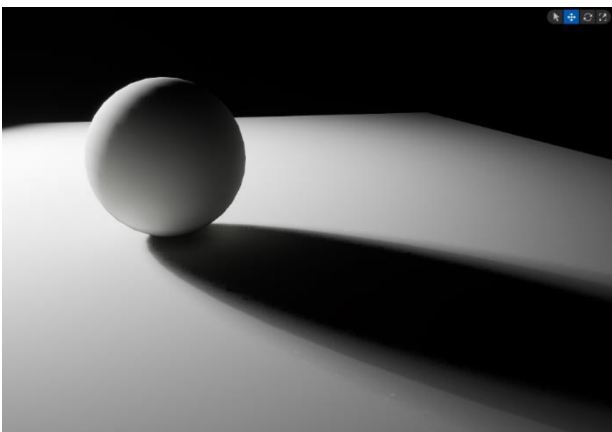


The following graph illustrates the order of raytracing visibility.

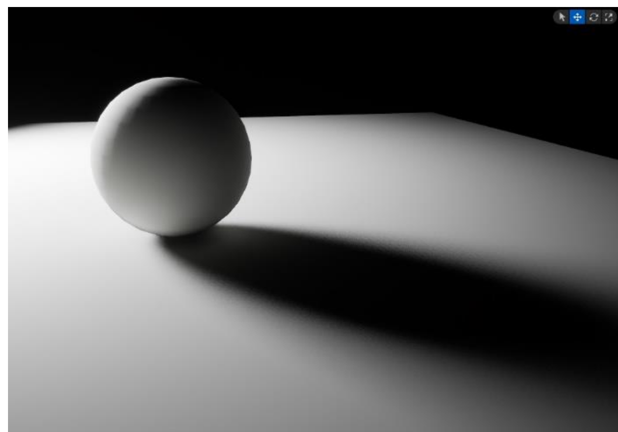


Raytraced Area Lights

One of the advantages of using RT shadows is to get accurate soft area light shadows from rectangular lights or other light types with large radius.

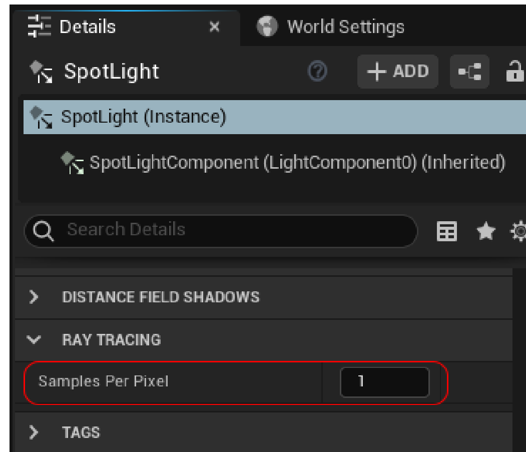


Area light shadow map



Area light ray traced shadow

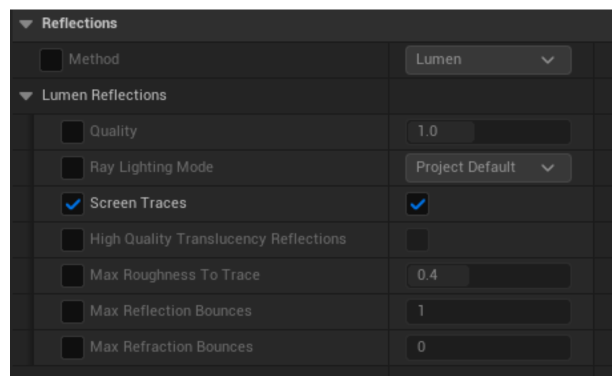
By default, the engine casts a ray per pixel which works for most cases. For more complex cases where quality is the priority, it is possible to increase the number of raytraced samples to reduce any noise. Keep in mind that this has direct impact on the GPU performance.



As mentioned earlier, the NVRTX branch includes Sampled Lights (RTXDI) which allows rendering area lights shadows at fixed GPU cost with area light shadows comparable to offline path-tracers.

Reflections

As covered previously, Lumen solves reflections differently from UE4 raytracing reflections. It does screen space traces when possible while preserving raytracing for off screen samples. Screen traces can be disabled from the Lumen Reflection options in Post Process Volume to completely depends on ray traced reflections:



Screen traces settings such as the trace distance and depth threshold are not exposed in Post Process Volume, these settings can be tweaked through the following CVars:

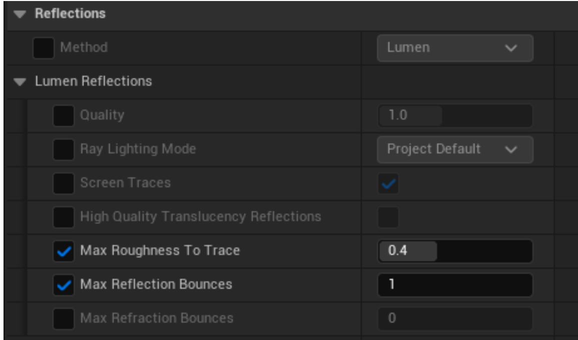
CVar	Description
`r.Lumen.Reflections.DistantScreenTraces`	Whether to do a linear screen trace starting where Lumen Scene ends to handle distant reflections.
`r.Lumen.Reflections.DistantScreenTraces.DepthThreshold`	Depth threshold for the linear screen traces done where other traces have missed.

r.Lumen.Reflections.DistantScreenTraces.MaxTraceDistance	Trace distance of distant screen traces.

Reflection Optimization

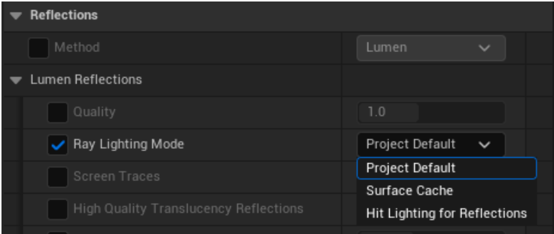
Similar to legacy raytraced reflections in UE4, HW Lumen reflections performance can be optimized by limiting some of the expensive calculations such as number of reflection bounces and/or skip casting rays for rougher surfaces.

These settings are exposed in the Post Process Volume as shown in the screenshot.

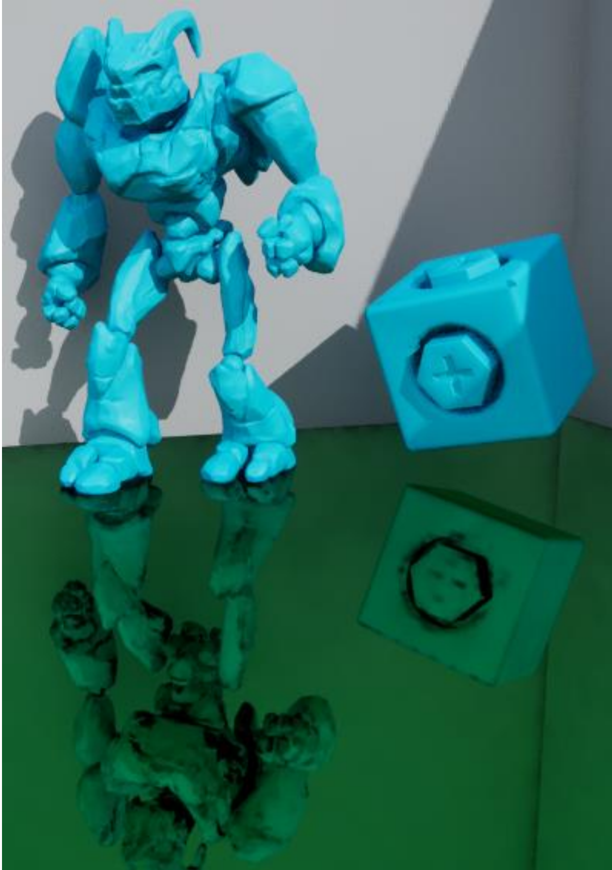


Accurate Chrome/Mirror Reflections

Most of the times raytracing reflections against the surface cache is sufficient, especially if the scene materials are mostly rough. For cases where accurate reflections are important, it is possible to switch raytracing from surface cache to evaluating materials at hit points. This can be enabled in the project settings as covered previously. However it can be overridden from the Post Process Volume for controlling the effect locally.



Using **Hit Lighting** mode impacts the GPU performance in favor of having accurate reflections.

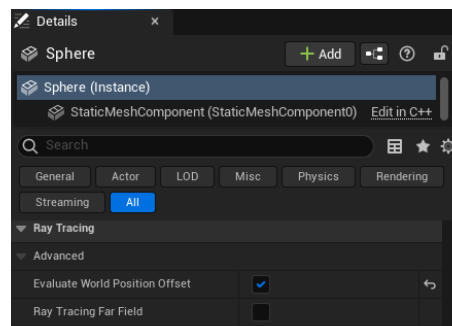


Lumen hardware raytracing mirrored reflections against Surface Cache

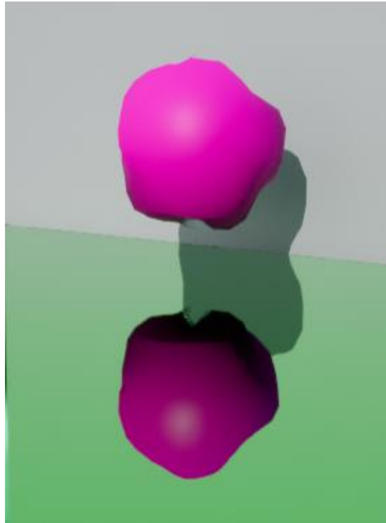
Lumen hardware raytracing mirrored reflections against Hit Lighting

Materials with World Position Offset

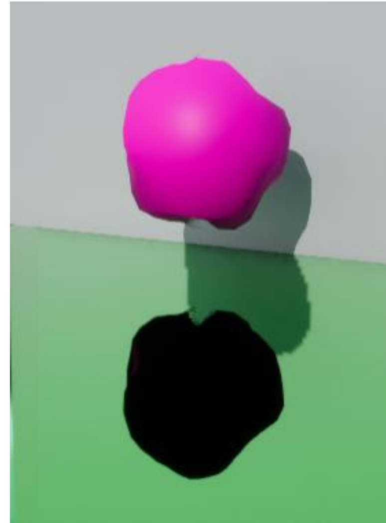
Generally materials with World Position Offset (WPO) are more challenging to render in raytracing as the deformation applied in the vertex shader need to be reflected in the raytracing acceleration structure. This causes additional performance cost to the BLAS updates so this effect is disabled by default for all meshes.



Overriding 'Evaluate World Position Offset' in the raytracing section will applies the deformation in secondary rays (i.e. reflections and GI) however with the surface cache mode, materials with WPO are not supported properly. To work around this current limitation, it is possible is to override the reflection mode to **Hit Lighting**.



Correct reflection of WPO material using Lumen 'Hit Point'



Black reflection of WPO material using default 'Surface Cache' mode of Lumen

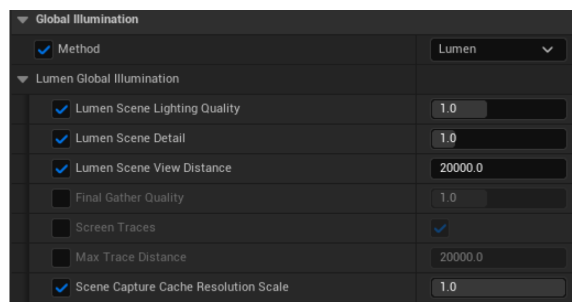
To keep the performance cost manageable in large scene, WPO only works for a short distance around the camera, by default 50 meters. This can be adjusted using the following CVar:

```
r.RayTracing.Geometry.StaticMeshes.WPO.CullingRadius
```

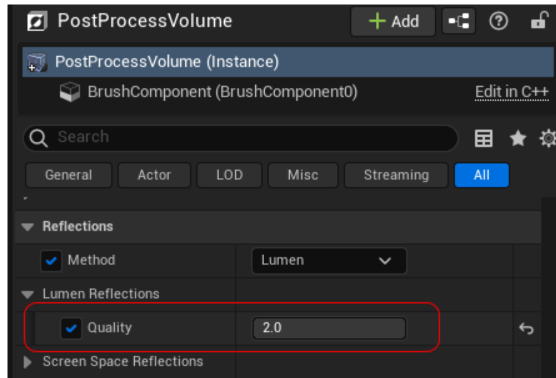
The NVRTX branch provides a feature for efficiently raytracing large number of instances with WPO. This allows ray tracing effects to scale well in large dynamic biomes with lots of vegetation.

Reflection Quality and Lumen Scene

As noticed previously, surface cache reflections are directly affected by the fidelity of **Lumen Scene** representation. There are a few settings exposed in the post process volume that can be tweaked to improve it. Most of the settings are grouped under the **Lumen Global Illumination** section. The group name might be misleading however these settings are not limited to Lumen GI but also impact the Lumen Surface Cache reflections since both depend on the simplified Lumen scene representation.



Under the **Lumen Reflection** section, there is one option (**Quality**) that can help in improving the sharpness of Lumen reflections (but not fidelity of Lumen scene representation).



Increasing any of the previous settings might help with quality of the surface cache reflections, however it increases the GPU cost.

Global Illumination

Several settings related to Lumen GI settings were already previously with Lumen Reflection. There are few additional settings in the post process volume for controlling the Final Gathering quality and the temporal amortization of the radiance cache. Increasing the Final Gather quality helps improving the accuracy and stability of GI for scenes with small details and emissive objects and the expense of higher GPU cost.

By default, UE utilizes screen probes for the final gathering. It is possible to fine tune its settings through the CVars: `r.Lumen.ScreenProbeGather.*`

In 5.4, additional final gather mode were introduced. One of the new modes leverage ReSTIR. This is still a prototype and can experimented with through the following CVars: `r.Lumen.ReSTIRGather.*`

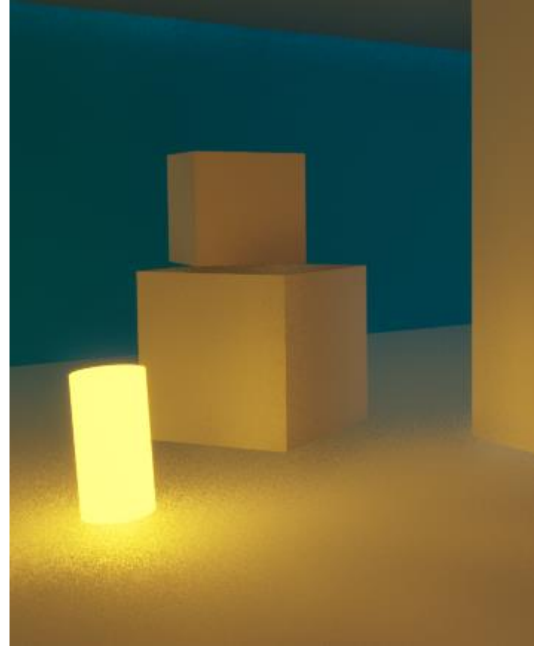
Although Lumen GI and reflections works closely together, it is possible to disable Lumen GI individually and keep only Lumen reflections. If Lumen reflections uses hit lighting, it is recommended also to disable the distance fields to save memory and performance as these are not required anymore neither by Lumen GI nor Lumen reflections.

Ambient Occlusion

Ambient Occlusion (AO) options still exist in the post process volume including raytraced ambient occlusion (RTAO). However in order to activate RTAO, Lumen needs to be disabled. With Lumen, Short Range AO is used instead. Short range AO is basically screen space directional occlusion which is useful for adding micro details around small areas that is hard to capture with the Lumen GI alone.

Short Range AO can be toggled (for debugging purposes) from the Show->Lumen->Short Range Ambient Occlusion.

Screen Space Direction Occlusion On (default)

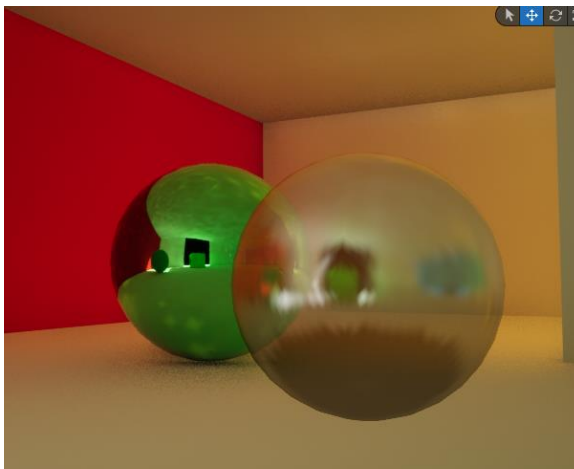


Screen Space Direction Occlusion Off

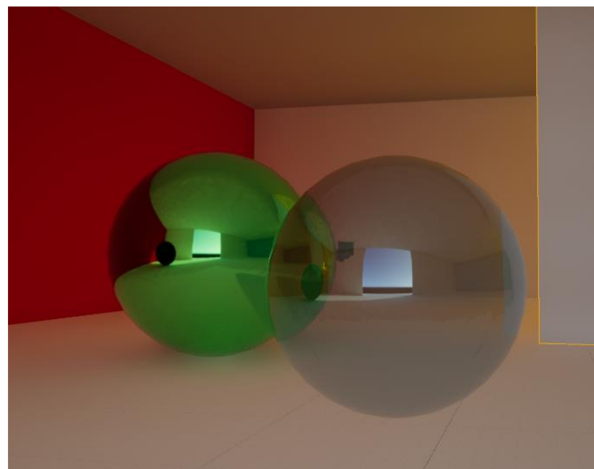
Translucency

Translucent materials are supported by Lumen GI and reflections. By default, Lumen will depend on the lower quality **Radiance Cache** to prioritize performance. Generally, this does not provide acceptable reflections specially for highly gloss translucent materials.

This can be improved by enabling the **High Quality Translucency Reflections** option either in the project settings or in the post process volume. This mode will render mirror reflections only for the first 'front' layer to the screen. As expected, this option incurs additional GPU cost when enabled.

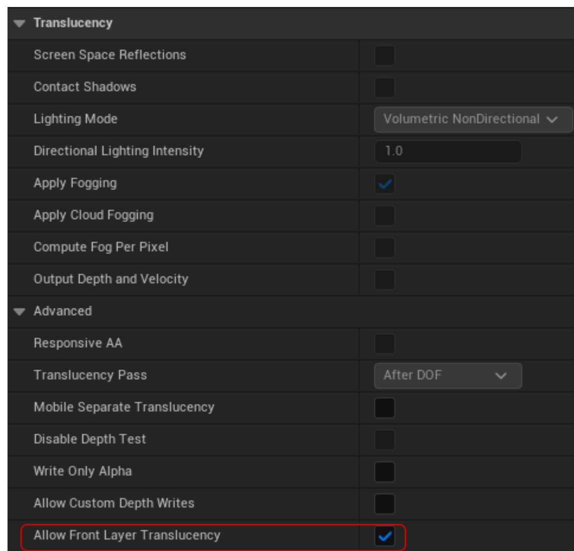


Translucent material using Radiance Cache for reflections



Translucent material with accurate reflections (front layer only)

By default, this option is applied to all translucent materials. However, it is possible to override it for a specific material if needed for performance reasons by unchecking the **Allow Front Layer Translucency** in the material editor.



In terms of refractions, as covered previously it is possible to leverage HW RT by enabling in the engine settings the **Ray Traced Translucent Refractions** in the project settings.

Raytracing Debugging and Visualization

In complex scenes it might be challenging to debug a certain raytracing issue. Several tools are available to aid in debugging raytracing in both editor and runtime development builds. This section will go through some of the important modes.

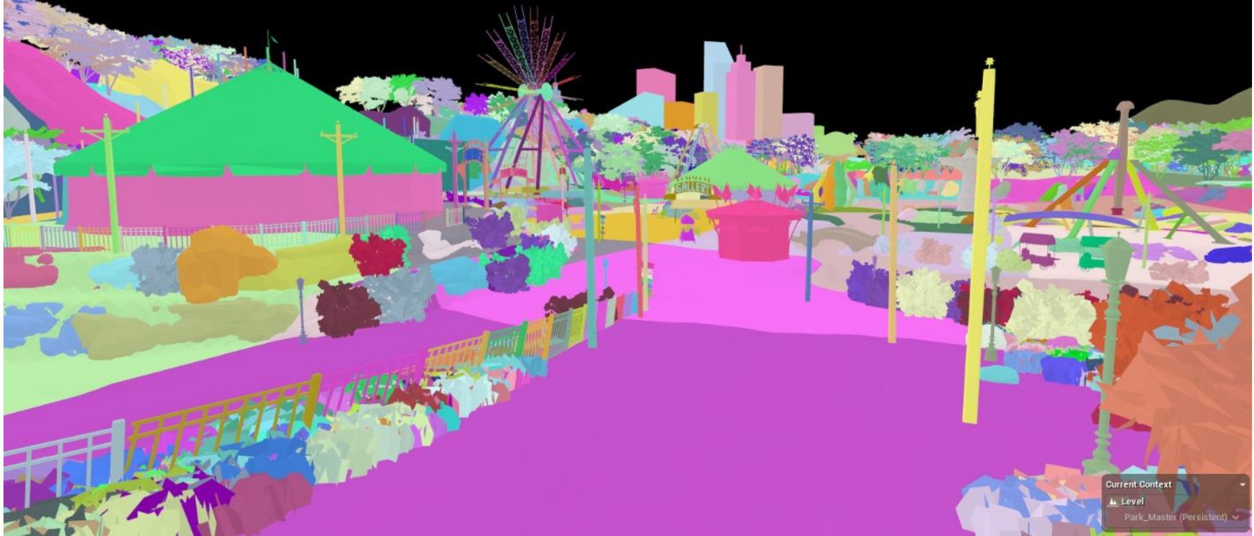
In editor, the **Ray Tracing Debug** menu provides several visualization modes of the different raytracing buffers and data structures. The same modes can be accessed in run-time through the CVar:

```
r.RayTracing.DebugVisualizationMode
```

The following will cover uses cases for some of the important visualization modes:

Instances

Instance count has direct impact on RT performance mainly due to the increased cost of TLAS updates. TLAS is rebuilt every frame, and has a cost on the Rendering Thread, RHI Thread, and the GPU. These costs are mostly proportional to how many mesh instances go into the acceleration structure. so, it is important to keep the instance count under control. This mode will colorize individual instances as passed to the RT structure to help in visually checking the complexity of the content.

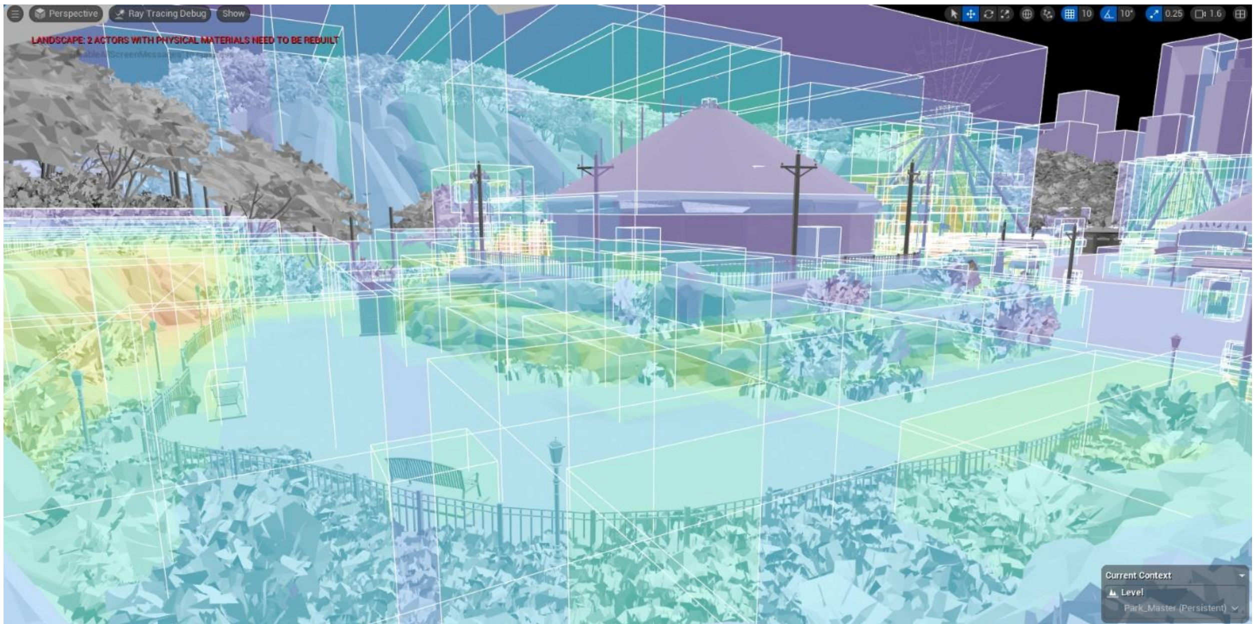


Instances Overlap

For each instance a bounding box (BBox) is calculated to be used in Bounding Volume Hierarchy (BVH) to accelerate the ray casting process.

Large overlapping between BBoxes increase the performance cost as a ray has to traverse more instances. This visualization mode shows the BBoxes for all instances and color code the amount of overlapping.

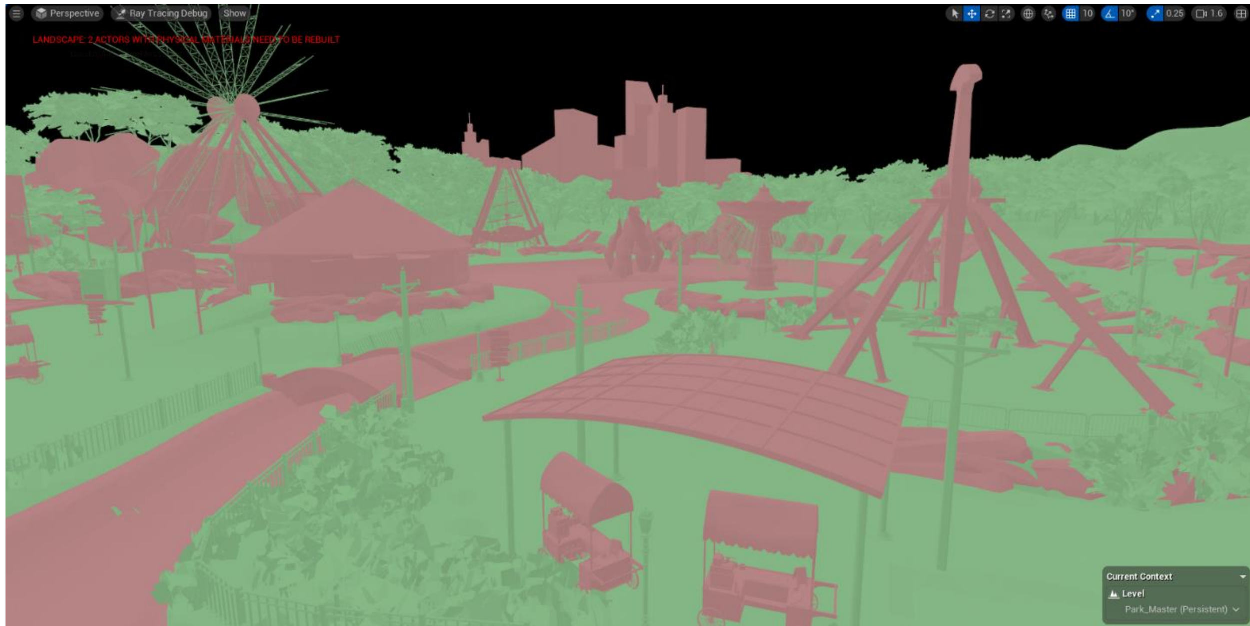
The tips and tricks section covers few examples and recommendations for minimizing BBox overlapping.



Dynamic instances

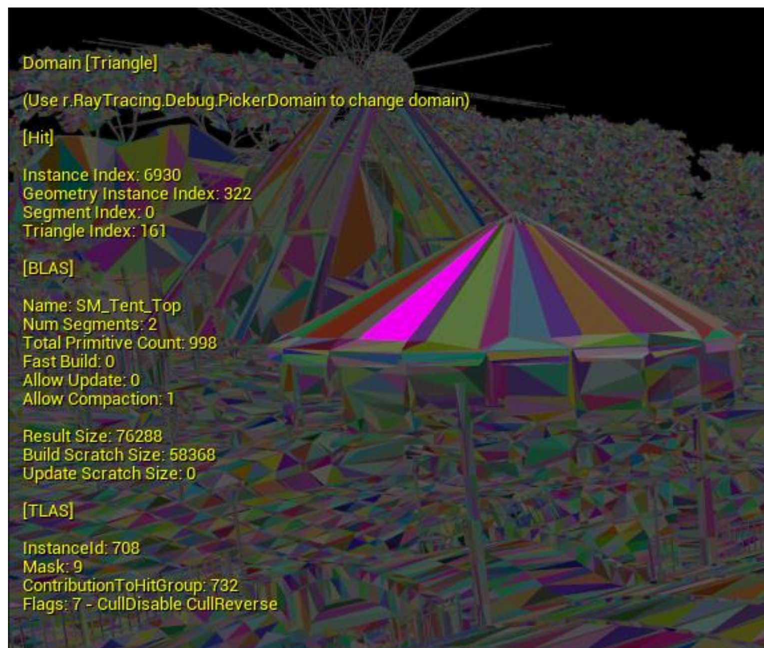
Dynamic instances where its geometry can deform such as Skeletal Meshes, Landscape morphing level of details, Niagara Particles, … etc have to rebuild their BLAS every frame. This impact RT performance negatively specially if these meshes have high triangles count.

This visualization mode help identifying all dynamic instances in the scene by color coding them in green.



Picker

This is an interactive debugging tool that allows querying any instance or triangle in the ray tracing scene for additional information. The Picker mode can operate on two different domains: Triangles and Instances. The mode can be set through the CVar: `r.RayTracing.Debug.PickerDomain`

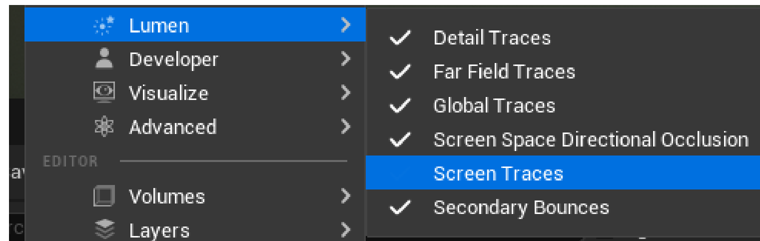


Lumen Debugging and Visualization

This section covers some of the visualization modes available for debugging different Lumen components.

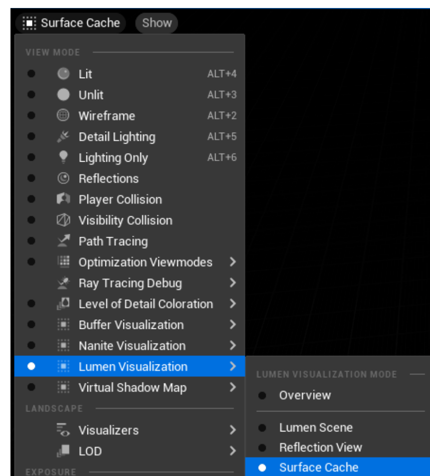
Screen Traces

Reflection screen traces have the highest priority as explained previously. So to isolate off screen reflections it is useful to be able to toggle screen traces either in editor through the **Show->Lumen->Screen Traces** or using CVar: `r.Lumen.Reflections.ScreenTraces`

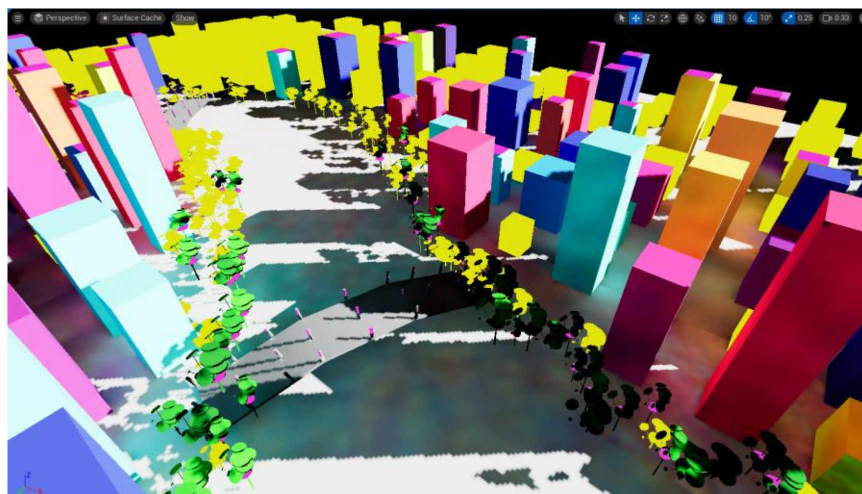


Surface Cache

Missing or invalid surface cache will cause issues with Lumen components that depend on it. To validate the surface cache, it is possible to visualize it in editor using the **Surface Cache** mode under the **Lumen Visualization** menu.



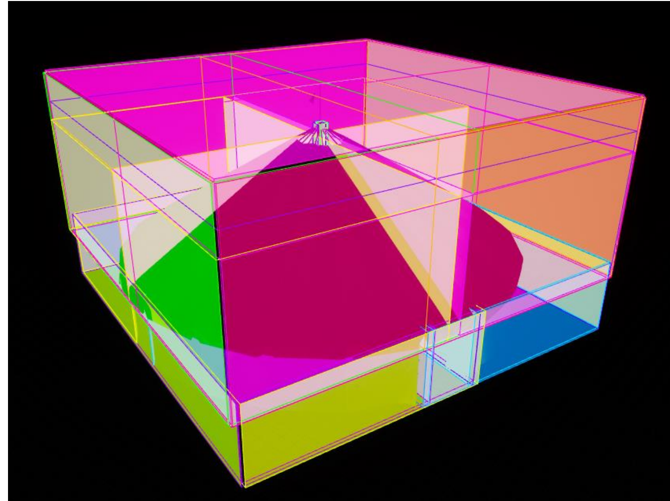
Ideally the scene should be covered properly by surface cache showing colors other than solid pink for missing surface cache coverage or yellow for culled.



In cases where a specific object is failing the surface cache, it is possible to visualize the card placements that are used for auto generating the surface cache to further debug the issue.

The following CVar enables the card placement visualization:

```
r.Lumen.Visualize.CardPlacement
```



Far Fields

When leveraging Far Fields in open world scenes, it is useful to visualize the generated far fields meshes that are used by Lumen for far objects. In editor, Far Field can be visualized using the ray tracing debug visualization mode for **FarField**. Or through CVars:

- `r.RayTracing.DebugVisualizationMode FarField`
- `Showflag.RayTracingDebug 1`

Ideally the far fields should be the same as the first level of the generated HLODs. In case this is not the case, this could be an issue related to HLODs.

For additional information about debugging Lumen components, it is recommended to refer to the official documentation: [Lumen Technical Details | Epic Developer Community (epicgames.com)]

Raytracing Performance Profiling

This section covers some of the performance stats available in the engine to get more detailed breakdown of the raytracing perf on different threads. For in-depth performance profiling it is recommended to use dedicated tools such as Unreal Insights.

STAT SceneRendering

This stat provides a breakdown for most of the tasks performed on the render thread (CPU) including raytracing related tasks. Regarding RT, the important regime to pay attention to is the **GatherRayTracingWorldInstances**. This is directly impacted by the instances count which is also listed under the counter section in the same stat.

Scene Rendering [STATGROUP_scenerendering] Cycle counters (flat)	CallCount	InclusiveAvg	InclusiveMax	ExclusiveAvg	ExclusiveMax
RenderViewFamily	1	12.59 ms	15.54 ms	2.21 ms	2.83 ms
GatherRayTracingWorldInstances	1	2.60 ms	3.60 ms	2.47 ms	3.43 ms
DeferredShadingSceneRenderer Lighting	1	1.25 ms	1.51 ms	0.81 ms	0.97 ms
InitViews	1	0.63 ms	0.78 ms	0.06 ms	0.09 ms
Lighting drawing	1	0.44 ms	0.59 ms	0.00 ms	0.00 ms
DeferredShadingSceneRenderer AfterBasePass	1	0.14 ms	0.18 ms	0.03 ms	0.04 ms
Water pass drawing	2	0.06 ms	0.08 ms	0.06 ms	0.08 ms
DeferredShadingSceneRenderer RenderLightShaftOcclusion	1	0.05 ms	0.07 ms	0.05 ms	0.07 ms
Bind ray tracing pipeline	2	0.05 ms	0.07 ms	0.05 ms	0.07 ms
Translucency drawing	1	0.05 ms	0.07 ms	0.03 ms	0.05 ms
Base pass drawing	1	0.04 ms	0.07 ms	0.04 ms	0.07 ms
DeferredShadingSceneRenderer RenderLightShaftBloom	1	0.04 ms	0.06 ms	0.04 ms	0.06 ms
InitViewsPossiblyAfterPrepass	1	0.03 ms	0.06 ms	0.02 ms	0.04 ms
DeferredShadingSceneRenderer RenderFinish	1	0.03 ms	0.04 ms	0.02 ms	0.02 ms
Dynamic shadow setup	2	0.03 ms	0.05 ms	0.01 ms	0.02 ms
Cache Uniform Expressions					
DeferredShadingSceneRenderer ViewExtensionPreRenderView	1	0.02 ms	0.03 ms	0.02 ms	0.03 ms
DeferredShadingSceneRenderer RenderSkyAtmosphere	1	0.02 ms	0.03 ms	0.02 ms	0.03 ms
DeferredShadingSceneRenderer Render Init	1	0.01 ms	0.04 ms	0.01 ms	0.04 ms
RenderVelocities	1	0.01 ms	0.02 ms	0.01 ms	0.02 ms
DeferredShadingSceneRenderer RenderFog	1	0.01 ms	0.01 ms	0.01 ms	0.01 ms
DeferredShadingSceneRenderer FXSystem PreRender	1	0.01 ms	0.01 ms	0.01 ms	0.01 ms
DeferredShadingSceneRenderer ViewExtensionPostRenderView	1	0.01 ms	0.01 ms	0.01 ms	0.01 ms
Wait RayTracing Add Mesh Batch	1	0.01 ms	0.10 ms	0.01 ms	0.03 ms
DeferredShadingSceneRenderer FGlobalDynamicVertexBuffer Commit	2	0.00 ms	0.01 ms	0.00 ms	0.01 ms
[2 more stats. Use the stats.MaxPerGroup CVar to increase the limit]					

Counters	Average	Max	Min
Ray tracing active instances	6,529.00	6,529.00	6,529.00
Ray tracing total instances	6,529.00	6,529.00	6,529.00
Decals in scene		336.00	336.00
Mesh draw calls	173.92	346.00	1.00
Present time	0.01 ms	0.02 ms	
Lights in scene		135.00	135.00
Decals in view	11.00	11.00	11.00
Lights using light shafts	2.00	2.00	2.00
Ray tracing pending build primitives		0.00	0.00
Ray tracing pending builds		0.00	0.00

STAT D3D12RayTracing

This stat focuses mainly on ray tracing costs on the RHI Thread (CPU) in addition to several useful counter and memory usages. With this stat it is possible to get more metrics about the performance and memory related to the update BLAS for dynamic objects.

D3D12RHI: Ray Tracing [STATGROUP_D3D12RayTracing] Cycle counters (flat)	CallCount	InclusiveAvg	InclusiveMax	ExclusiveAvg	ExclusiveMax
BuildBottomLevel	1	0.63 ms	1.32 ms	0.63 ms	1.32 ms
SetRayTracingBindings	3	0.78 ms	1.65 ms	0.00 ms	0.00 ms
DispatchRays	23	0.75 ms	1.57 ms	0.75 ms	1.57 ms
BuildTopLevel	1	0.18 ms	0.38 ms	0.18 ms	0.38 ms
CreateShaderTable	2	0.06 ms	0.14 ms	0.06 ms	0.14 ms

Memory Counters	UsedMax	Mem%	MemPool	Pool Capacity
Total Used Video Memory	233.13 MB		Physical	
Total BL AS Memory	201.13 MB		Physical	
Dynamic BL AS Memory	133.00 MB		Physical	
Static BL AS Memory	68.13 MB		Physical	
TL AS Memory	32.00 MB		Physical	

Counters	Average	Max	Min
Triangles in all BL acceleration structures		4,344,656.00	4,341,443.50
Allocated view descriptors		2,500,000.00	2,500,000.00
Allocated sampler descriptors		20,480.00	20,480.00
Used view descriptors (per frame)	3,118.35	5,559.00	1,030.00
Allocated bottom level acceleration structures		351.00	345.00
Compiled shaders (total)		310.00	310.00
Created pipelines (total)		309.00	309.00
Updated BL AS (per frame)	106.00	212.00	106.00
Max used sampler descriptors in a single heap	54.13	56.00	56.00
Used sampler descriptors (per frame)	46.00	89.00	6.00
Allocated top level acceleration structures		12.00	11.00
Allocated sampler descriptor heaps		10.00	10.00
Allocated view descriptor heaps		10.00	10.00
Built BL AS (per frame)	6.00	12.00	6.00
Built TL AS (per frame)	3.00	6.00	3.00

STAT GPU

Most GPU tasks are listed in this stat. Raytracing related costs are mainly allocated under the following regimes:

- RayTracingScene: Includes TLAS updates.
- RayTracingGeometry: Includes BLAS builds for the dynamic geometry.

- RayTracingPrimaryRays: This marks the cost of ray tracing primary ray for translucency objects.
- RayTracingTranslucency: This wraps the translucency pass cost, but it excludes the inner cost of the RayTracingPrimaryRays.

Additional ray tracing costs are included in:

- LumenReflection: This includes the RT cost when enabled for reflections. It could be either the hit lighting cost or surface cache depending on the project settings.
- LumenSceneLighting: This is mainly for Lumen Scene updating. It includes some of the RT cost when enabled for GI.

GPU [STATGROUP_gpu]

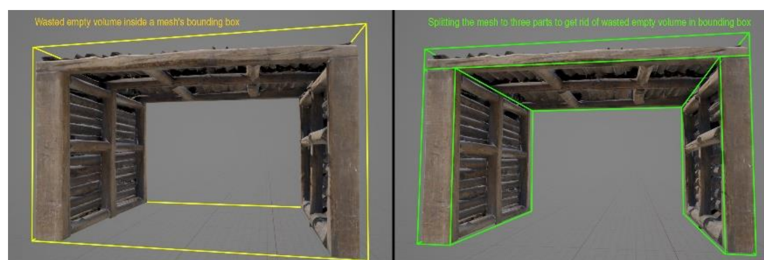
Counters	Average	Max	Min
[TOTAL]	16.05	19.08	13.52
Basepass	3.38	4.12	3.09
Lights	1.47	1.96	1.30
LumenReflections	1.29	1.75	1.08
RayTracingPrimaryRays	1.17	1.42	1.03
RayTracingGeometry	0.30	0.34	0.26
RayTracingScene	0.25	0.29	0.24
DistanceFields	0.28	0.61	0.10
SkinnedGeometryUpdateBLAS	0.02	0.03	0.01
Fog	0.02	0.03	0.01
VolCloudReconstruction	0.01	0.02	0.01
GPUSkinCache	0.01	0.02	0.01
Ray Tracing Translucency	0.02	0.05	0.01
VisibilityCommands	0.01	0.01	0.01
VolCloudComposeOverScene	0.01	0.01	0.01
SkyAtmosphere	0.01	0.01	0.01
RayTracingUpdate	0.00	0.00	0.00
FXSystemPreRender	0.00	0.00	0.00
EndOfFrameUpdates	0.00	0.00	0.00

Tips and Tricks

With most of the essential UE5 ray tracing systems covered, this section introduces several tips and tricks related to content creation to have the best performance out or raytracing in UE5.

Geometry Complexity

A complex mesh with excessive wasted volume in its bounding box can impact ray tracing performance negatively. Consider splitting the mesh to achieve tighter bounding boxes for optimal raytracing performance.



Separating complex meshes not only helps BLAS but can benefit culling and streaming. However, increasing the number of objects can negatively impact TLAS. Striking a balance between BLAS and TLAS is key for optimal performance.

Kit Bashing

Kit bashing is a common approach to assemble levels. This approach becomes more popular with the wide availability of marketplace and Quixel assets. So with this approach most of the assets are generic/non-original assets that are not created specifically for the assembled scene.

Because such assets are not created to fit the level design, it causes high overlap between assets which increases the BVH complexity, leading to longer traversal times and slower performance.

For large intersections it might be recommended to create custom mesh that better fits the level layout. If creating a custom mesh is not possible and the mesh is mostly hidden under multiple layers of intersecting meshes consider disabling ray tracing for it.

Sky Boxes

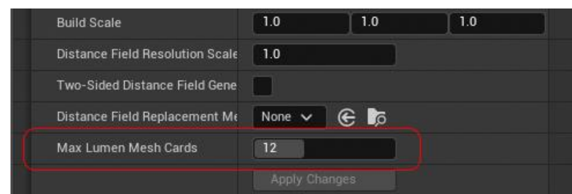
As mentioned earlier, HW RT performance impacted negatively by excessive meshes overlap. Large meshes that overlap the entire scene are a performance issue, such as a skybox. These meshes should have their ray tracing visibility disabled.

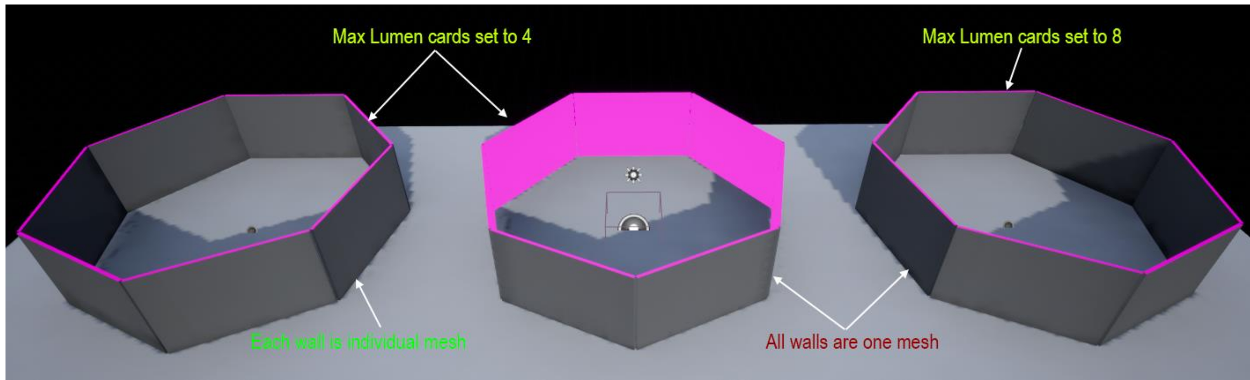
Lumen Surface Cache

Lumen generates the surface cache automatically by parameterizing the scene using cards. Cards count and placement depends on the mesh shape. Thus, it is recommended to build the mesh from relatively simple shapes and avoid complex merged meshes. For example, instead of creating an interior building from one complex mesh with all walls and ceiling merged. It is recommended to assemble it in UE5 from individual modular walls. This modular approach is also beneficial to other areas:

- Memory (due to better instancing opportunities)
- Raytracing performance (due to potentially better acceleration structure).

In cases where complex meshes are inevitable, it is possible to increase the number of Lumen cards to minimize the surface cache missing coverage. This can be done for each asset from the static mesh editor:





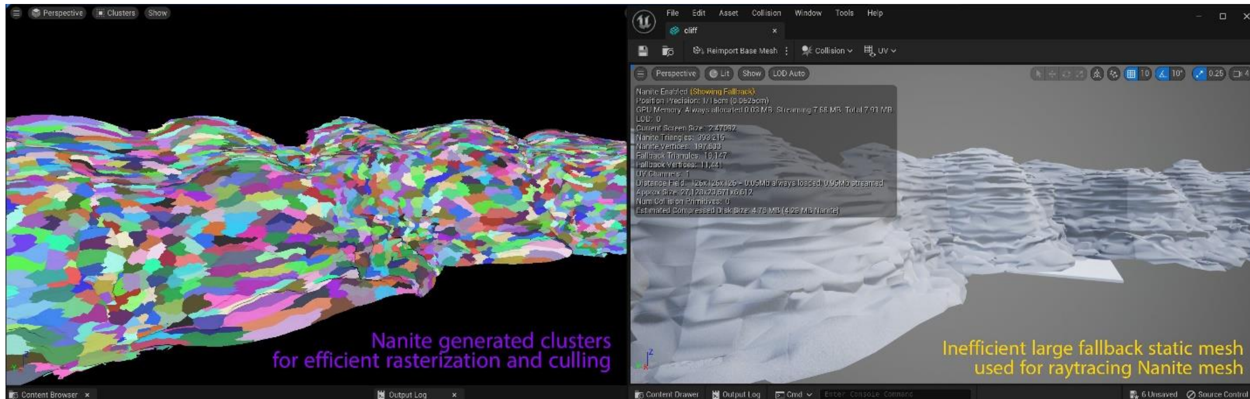
Surface cache coverage for different content build approach and Lumen cards count

HLODs and Lumen Far Fields

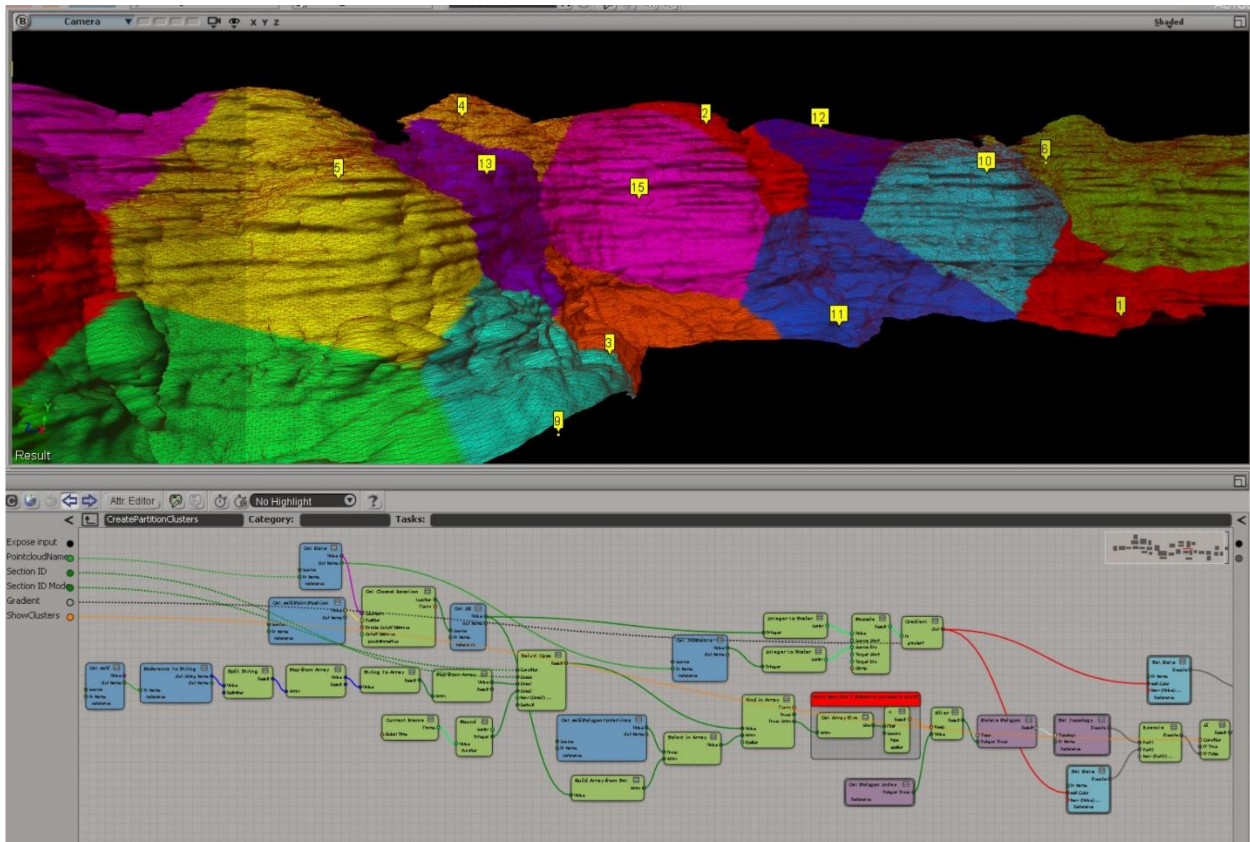
In addition to culling raytracing instances, one approach to minimize the processed instances in large open world scenes is to couple culling with HLODs and Far Field to optimize and scale down Lumen Hardware Ray Tracing performance.

Nanite Meshes

While Nanite can render large, dense meshes without manual optimization, it falls back to a static mesh for raytracing. Having the fallback mesh as one large dense mesh can negatively impacts raytracing. To maintain Nanite RT efficiency, consider breaking large meshes (like large detailed mountains) into smaller chunks and tweak their fallback mesh triangle count.



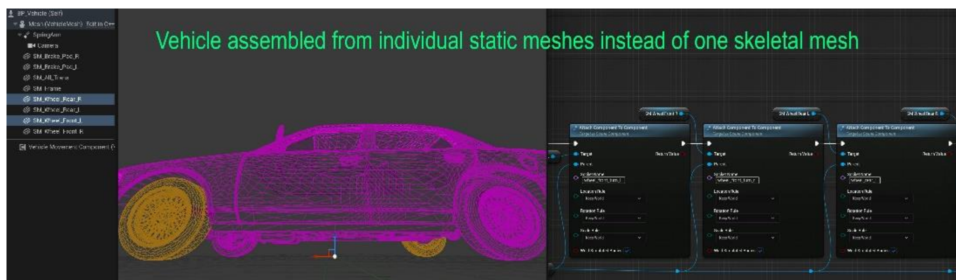
This process can be automated in UE using geometry scripting or using an external DCC as shown in the screenshot below.



Skeletal Meshes

As mentioned earlier, skeletal meshes are one of the dynamic mesh types that requires rebuilding their BLAS every frame which can incur significant cost. The BLAS rebuilds are proportional to the total number of triangles. To minimize the cost of BLAS rebuilds, it is possible to use a lower LoD (with significantly less triangles) for ray tracing instead of LoD0 that has the highest triangle count.

Another approach is to avoid unnecessary skeletal meshes such as skinning non-deformable objects like vehicles. Ideally non-deformable vehicles should be assembled in-engine from static meshes for more efficient raytracing and animation.

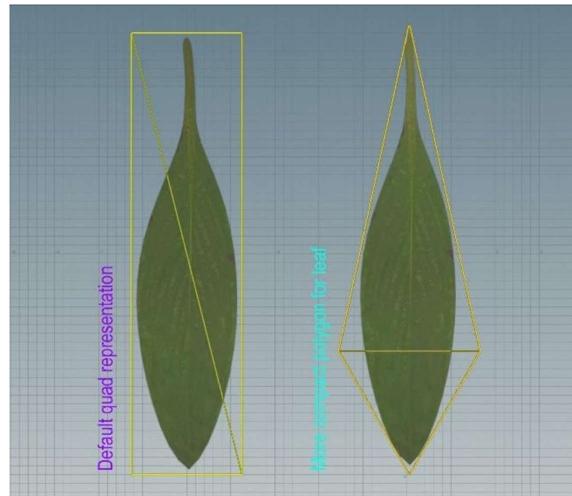


For further optimization, merge vehicle meshes into a single static mesh for distant LoDs where animation details are not noticeable. This can improve scalability and performance in your open-world game.

Masked Materials

In ray tracing, masked materials have more expensive shader evaluation than opaque materials due the invocation of any-hit shader that interrupts hardware intersection search.

For scenes with lots of vegetation this might add up and show negative impact on ray tracing performance on the GPU. When possible, minimizing the area not marked as opaque is a simple way to increase performance. Using more triangles to define the non-opaque area more accurately is likely a good trade-off, however in some cases it is possible to achieve this without increasing triangle by just modifying the topology as shown in the example below:



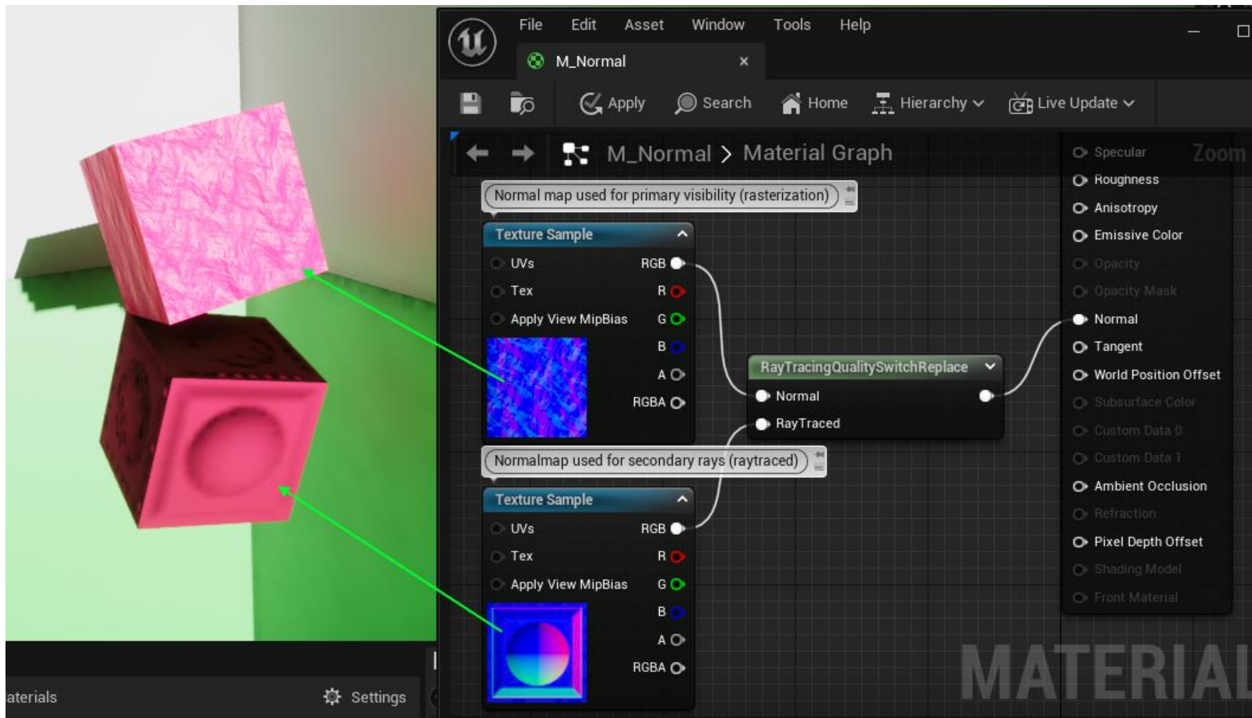
In the NVRTX branch, meshes with masked materials can leverage the new **Opacity Micro Maps** to encode the mask opacity in micro-triangles and makes it possible to trace rays at high performance without the need for the shader invocations.

Complex Materials

The material editor in UE5 allows creating complex effects which might not be practical to evaluate in ray tracing effects for both performance and visual reasons.

It is possible to branch the shader logic for raytracing effects using the **Ray Tracing Quality Switch Replace** node. This node allows materials to use a simpler/different version of themselves in HWRT while keeping the complex shader for primary rasterization thus dramatically reducing HWRT cost. This should be set up in the master materials so that all material instances can benefit from this inherited setting for maximum efficiency.

Additionally, this node can be used for reducing noise in raytracing due to high frequency procedural textures. One example is a material using detailed normal map, where the raytracing quality switch can provide flat normal map for secondary rays.



Lumen hardware raytracing reflections with screen traces set to off

Since Lumen is solving reflections in UE5, this node behaves a bit differently from UE4. For solving reflections, Lumen prioritizes screen-space tracing over hardware raytracing. As a result, materials using this node won't work at the screen-space tracing phase. The node only works with off-screen hardware tracing.