Performance Trade-off Analysis of Nanite and Traditional LOD Systems in Dynamic Open-World Environments Using Unreal Engine 5.7

Jeffrin A
Computer Science and Engineering
RMK Engineering College
240132.cs@rmkec.ac.in

Latheshkumar S
Computer Science and Engineering
RMK Engineering College
240938.cs@rmkec.ac.in

Madesh S
Computer Science and Engineering
RMK Engineering College
240087.cs@rmkec.ac.in

Kamalesh Balaji D N
Computer Science and Engineering
RMK Engineering College
240674.cs@rmkec.ac.in

T Sethukarasi
Computer Science and Engineering
RMK Engineering College
tsk.cse@rmkec.ac.in

N Banupriya
Computer Science and Engineering
RMK Engineering College
nbp.cse@rmkec.ac.in

***Abstract*—This paper investigates the real-time performance trade-offs between Unreal Engine 5.7's Nanite virtualized geometry system and traditional Level of Detail (LOD) pipelines in dynamic open-world environments containing animated foliage. While traditional LOD methods reduce geometric complexity, they introduce CPU draw-call overhead and visible LOD transitions. Nanite addresses these limitations through virtualized micro-polygon rendering, but its efficiency in foliage-heavy scenes with alpha-masked materials and vertex animation remains unclear.**

To analyze this behavior, we perform a controlled experimental comparison of Nanite and LOD pipelines under identical scene conditions with wind-driven vegetation. Performance is evaluated on an RTX 4060 system using GPU profiling metrics including frame time, thread execution costs, draw calls, primitive counts, and VRAM usage. Visual diagnostics such as shader complexity heatmaps further illustrate increased pixel shading cost in dense foliage regions. Results show that despite significant reductions in draw calls and primitives, traditional LOD achieves higher frame rates in foliage-dense scenes. These findings highlight the necessity of hybrid geometry optimization strategies for modern open-world game development.

***Index Terms*—Unreal Engine 5, Nanite, Level of Detail, Real-time Rendering, GPU Profiling, Open-world Optimization, Virtualized Geometry**

I. Introduction

Modern open-world games require the rendering of extremely dense geometric content while maintaining real-time performance. This demand makes geometry optimization a critical component of real-time rendering pipelines. For decades, rendering engines have been fundamentally bottlenecked by the CPU-driven draw call overhead associated with rendering individual discrete meshes via APIs like DirectX and Vulkan. Traditional Level of Detail (LOD) techniques reduce this geometric complexity by switching between precomputed mesh resolutions based on camera distance or screen-space error, thereby improving rendering efficiency in large-scale environments [1].

Unreal Engine 5 introduces Nanite, a disruptive virtualized geometry system that enables the rendering of film-quality assets without manual LOD creation. Nanite dynamically streams mesh clusters and renders them at pixel-scale detail. By decoupling the geometry complexity from the CPU and utilizing a custom software rasterizer for micro-polygons, Nanite significantly reduces draw calls and eliminates visible LOD transitions, enabling an unprecedented level of static detail [2].

Despite these advantages, traditional LOD systems are still widely used in foliage-heavy scenes due to their compatibility with vertex animation and lower runtime overhead. When dealing with deformable geometry, such as wind-animated trees and grass, the assumptions that Nanite relies upon for its efficiency begin to break down. Recent experimental studies evaluating Nanite have primarily focused on virtual reality or static architectural environments, where forward shading and controlled camera motion dominate the rendering workload [3]. The RTX 4060 represents a widely used mid-range target platform for modern PC game development, making these findings directly relevant to real production constraints. However, dynamic open-world environments with animated vegetation present entirely different performance characteristics that remain insufficiently explored.

The main contributions of this paper are:

- A real-time performance comparison of Nanite and traditional LOD systems in a dynamic open-world environment.
- Fine-grained GPU pipeline analysis using Unreal Engine profiling tools.
- Memory and geometry complexity evaluation on RTX-class hardware.
- Practical guidelines for selecting geometry optimization techniques for modern game production.

II. Problem Statement and Contributions

A. Problem Statement

Real-time open-world rendering requires balancing geometric complexity, memory usage, and GPU processing cost. Nanite introduces a virtualized geometry pipeline that eliminates manual LOD generation and drastically reduces draw calls, effectively shifting the bottleneck from the CPU to the GPU. However, its performance in dynamic environments containing highly deformable vegetation remains unclear.

In foliage-heavy scenes, vertex animation through World Position Offset (WPO) modifies mesh topology every frame. This dynamic modification potentially reduces the efficiency of Nanite's hierarchical cluster culling, forcing the engine to conservatively render more clusters than strictly necessary. Traditional LOD systems, although geometrically heavier in terms of primitive counts, operate on precomputed mesh resolutions and may therefore achieve lower runtime GPU cost when heavily animated.

This raises an important research question: *Does Nanite provide better real-time performance than traditional LOD systems in dynamic open-world environments characterized by heavy vertex deformation?*

B. Contributions

The main contributions of this work are:

- A controlled experimental comparison between Nanite and traditional LOD pipelines in Unreal Engine 5.7.
- A detailed GPU-bound performance analysis using `stat GPU` profiling to isolate rendering bottlenecks.
- A comprehensive study of the relationship between draw calls, primitive count, VRAM usage, and total frame time.
- A decision matrix and practical guidelines for selecting geometry optimization techniques in open-world game production.

III. Related Work

A. Traditional LOD and Terrain Optimization

Traditional LOD techniques have been extensively used to reduce triangle count and improve rendering performance in large-scale terrains and vegetation-heavy environments. By dynamically selecting mesh resolution based on screen-space error, LOD systems significantly lower geometric complexity while maintaining visual fidelity [1]. Adaptive multiresolution mesh approaches demonstrate similar concepts by enabling view-dependent geometric refinement at runtime, optimizing the balance between detail and performance [4].

B. Virtualized Geometry and Nanite

Nanite introduces a virtualized geometry pipeline that replaces discrete LOD transitions with continuous mesh streaming. It renders only visible mesh clusters at the required pixel density, thereby drastically reducing draw calls and CPU overhead [2]. Nanite's architecture represents a shift toward compute-driven software rasterization, decoupling scene complexity from object count. The educational and developmental impact of this disruptive technology has been noted to significantly alter traditional 3D asset creation workflows, allowing artists to bypass traditional retopology and baking phases [5], [6].

C. Cross-Engine and VR Performance Analysis

Recent comparative studies have evaluated rendering optimization methods across different engines, noting that while Unity relies heavily on Data-Oriented Technology Stack (DOTS) and traditional LODs, Unreal Engine utilizes Nanite for massive static scaling [3], [7]. Experimental performance evaluations demonstrate that although Nanite reduces triangle count and draw calls, its GPU compute cost can increase depending on scene composition, leading to lower frame rates in high-resolution scenarios [8]. However, these studies are largely limited to static or controlled scenarios and do not systematically evaluate the impact of heavy vertex animation in dynamic open-world foliage environments.

IV. Nanite and LOD Rendering Pipeline Architecture

A. Traditional LOD Pipeline

The traditional Level of Detail system uses multiple precomputed mesh resolutions (e.g., LOD0 to LOD3). At runtime, the appropriate LOD level is selected based on camera distance and screen size. This reduces vertex processing cost and memory bandwidth while maintaining visual fidelity at different viewing ranges. However, this approach introduces LOD popping artifacts during transitions. Furthermore, it inherently increases draw calls because each mesh instance (or instanced group) must be evaluated and submitted to the GPU via the CPU draw thread.

B. Nanite Virtualized Geometry

Nanite replaces discrete LODs with a virtualized geometry system that operates on a Directed Acyclic Graph (DAG) of hierarchical mesh clusters. Instead of swapping entire meshes, Nanite processes geometry at the cluster level (typically 128 triangles per cluster). Only visible clusters at the required pixel resolution are streamed and rasterized.

The Nanite rendering pipeline introduces several advanced stages:

- **Cluster Culling:** Utilizing compute shaders, Nanite evaluates bounding boxes and pixel-space error metrics to determine visibility. This completely offloads frustum and occlusion culling from the CPU.
- **Visibility Buffer (VisBuffer):** Unlike traditional G-Buffers, the VisBuffer stores only the `Depth`, `MeshID`, and `PrimitiveID` for each pixel. This massively reduces overdraw overhead by decoupling geometry processing from material evaluation.
- **Dual Rasterization:** Nanite dynamically chooses between a custom software rasterizer for micro-polygons (triangles smaller than a pixel) and the hardware rasterizer for larger triangles.

- **Material Resolve:** Once the VisBuffer is complete, a full-screen pass reads the IDs and evaluates the actual materials only for the visible pixels.

C. Impact of Vertex Animation

Nanite's hierarchical DAG is heavily optimized for rigid, static geometry. In the presence of wind animation (WPO), vertex deformation modifies the spatial bounds of the clusters. This loss of cluster coherence forces the engine to conservatively estimate bounds, increasing the number of active clusters evaluated per frame and heavily inflating the GPU compute cost during the visibility buffer generation.

V. Methodology

A. Hardware and Software Configuration

All experiments were conducted on a desktop system equipped with an Intel Core i7 14th Generation CPU, NVIDIA GeForce RTX 4060 GPU with 8 GB VRAM, and 16 GB system memory. Unreal Engine 5.7 was used with the DirectX 12 rendering API. The application was rendered at a resolution of 1217×916. Vertical synchronization (VSync) and dynamic resolution scaling were disabled during profiling to obtain raw, uncapped frame rate measurements.

TABLE I: System Configuration

Hardware/Software	Specification
CPU	Intel Core i7 14th Gen
GPU	NVIDIA RTX 4060 (8GB VRAM)
System Memory	16 GB DDR5
Engine Version	Unreal Engine 5.7
Graphics API	DirectX 12 Ultimate

B. Test Scene Description

The experimental scene is a dynamic open-world environment containing dense vegetation (trees, grass, bushes) with wind animation implemented through World Position Offset. Two geometry pipelines were systematically evaluated:

- **Configuration A:** All scene geometry strictly utilizing Nanite-enabled meshes with preserve area enabled for foliage.
- **Configuration B:** Traditional static meshes utilizing manually configured 4-stage LOD chains and hierarchical instanced static meshes (HISM).

A fixed cinematic camera path was utilized for both configurations to ensure identical rendering conditions, frustum culling behavior, and temporal consistency.

C. Performance Metrics

Performance data was collected using Unreal Engine's internal `stat unit` (for thread-level bottlenecks) and `stat GPU` (for specific render pass costs) profiling tools. Metrics recorded include:

- Frames per second (FPS) and Total Frame Time (ms)
- Thread times (Game, Draw, RHI, GPU)
- Draw calls and total Primitive count
- GPU memory allocation (VRAM)
- Specific GPU pass timing (BasePass, Shadows, Lighting)

D. Scalability Settings

Lumen global illumination and virtual shadow maps were enabled to reflect modern Unreal Engine production settings. All scalability settings were fixed to **Medium** for both configurations to ensure identical shading cost. The frame rate was capped at 60 FPS to eliminate unnecessary GPU overproduction and stabilize frame-time measurements during profiling.

E. Performance Metrics Formulation

The relationship between frame rate and frame time is defined as:

$$T_{frame} = \frac{1000}{FPS} \tag{1}$$

where T_{frame} is measured in milliseconds.

The percentage performance variation between the Nanite and LOD pipelines is computed as:

$$\%\Delta P = \frac{P_{Nanite} - P_{LOD}}{P_{LOD}} \times 100 \tag{2}$$

where P represents the measured performance metric.

A frame is considered GPU-bound when:

$$T_{GPU} > T_{Game} \wedge T_{GPU} > T_{Draw} \tag{3}$$

The geometry reduction ratio is defined as:

$$R_{geo} = \left(1 - \frac{G_{Nanite}}{G_{LOD}}\right) \times 100 \tag{4}$$

where G denotes draw calls or primitive count.

The memory overhead introduced by Nanite is computed as:

$$M_{overhead} = \frac{M_{Nanite} - M_{LOD}}{M_{LOD}} \times 100 \tag{5}$$

Finally, the total frame time is governed by the slowest pipeline stage:

$$T_{frame} = \max(T_{Game}, T_{Draw}, T_{GPU}) \tag{6}$$

VI. Results and Analysis

A. Frame Rate and Thread Analysis

The traditional LOD pipeline achieved an average frame rate of 60.01 FPS, whereas the Nanite configuration achieved 43.81 FPS, corresponding to a performance decrease of 27.0%. As shown in the profiling metrics, the higher frame rate of the LOD pipeline indicates that the scene is primarily GPU-bound, since the configuration with significantly higher draw calls still performs better. Since the frame rate was capped at 60 FPS, the LOD configuration reaches the imposed upper limit. Therefore, its true peak performance is higher than the reported value, and the measured frame time represents a stable, capped workload rather than a hardware limitation.

TABLE II: Comprehensive Performance Trade-off Comparison Between Nanite and Traditional LOD

Pipeline Architecture	Average FPS	GPU Time (ms)	Draw Calls	Primitives	VRAM Usage	Shadow Cost (ms)
Nanite Virtualized Pipeline	43.81	21.23	518	24.8K	7.20 GB	5.18
Traditional LOD Pipeline	60.01	16.64	2734	539.5K	5.04 GB	3.39
Performance Delta	**-27.0%**	**+27.6%**	**-81.1%**	**-95.4%**	**+42.9%**	**+52.8%**

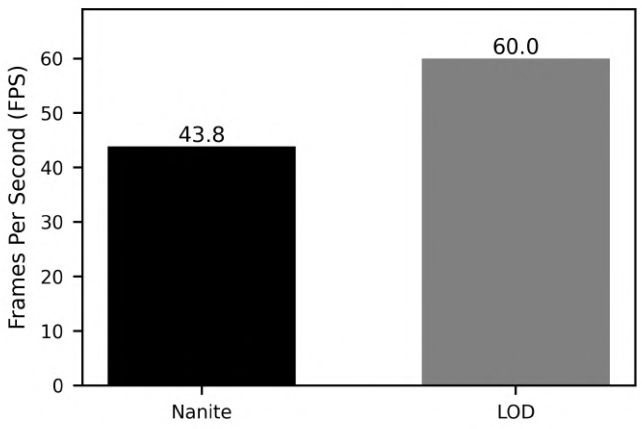


Fig. 1: Average frame rate comparison between Nanite and traditional LOD pipelines.

The game thread time decreased slightly from 7.16 ms to 6.59 ms (-7.9%), indicating that gameplay logic was not the bottleneck. However, the draw thread surprisingly increased from 6.51 ms to 23.00 ms (+253%) under Nanite. The GPU time increased from 16.64 ms to 21.23 ms (+27.6%), confirming the GPU bottleneck. (Fig. 2).

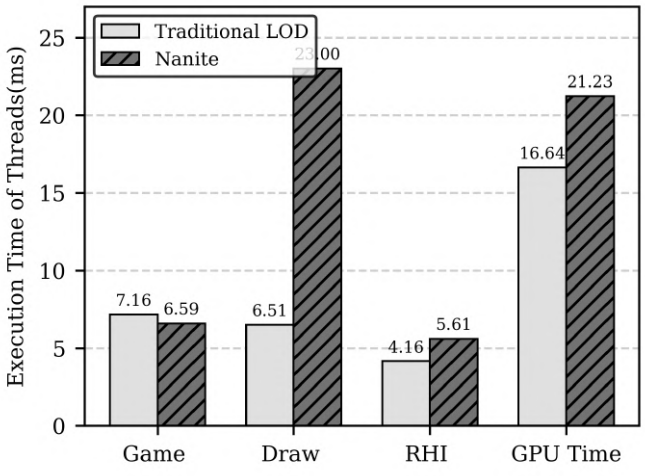


Fig. 2: Thread-level frame time breakdown for game, draw, RHI, and GPU threads.

B. Geometry Complexity vs. Render Cost

Nanite performed exceptionally well at its primary goal: reducing geometry submission. It reduced draw calls from 2734 to 518 (-81.1%) and primitive count from 539.5K to 24.8K (-95.4%). However, this 95% reduction in processed primitives inversely correlated with frame rate, proving that triangle count is no longer the primary indicator of performance in modern engines.

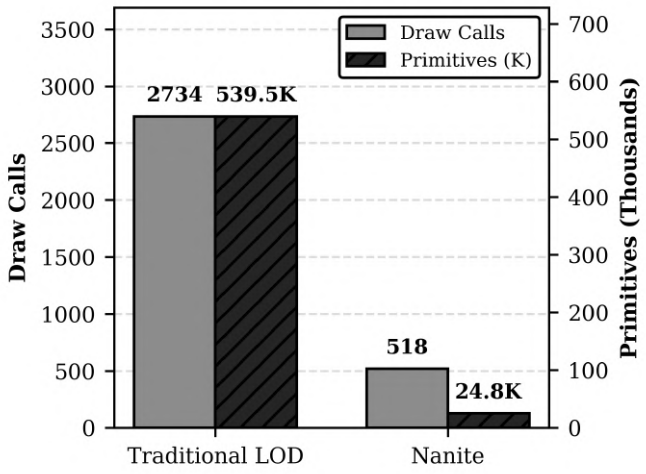


Fig. 3: Comparison of draw calls and primitive count for Nanite and traditional LOD.

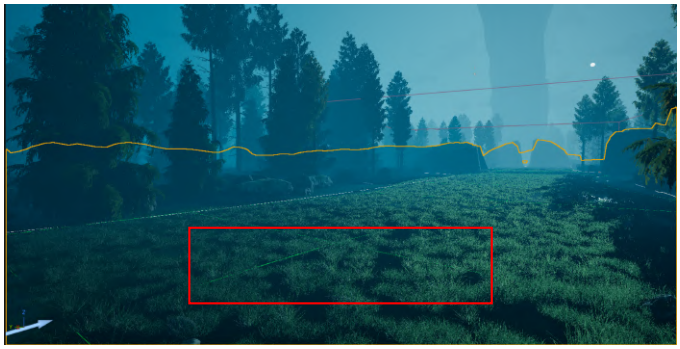
C. Visual Fidelity Observation

While both configurations produced visually comparable results at a macro level, detailed examination of specific environmental scenes reveals significant rendering discrepancies under dynamic conditions.

In Scene 1, the traditional LOD system maintains a dense, clustered distribution of grass meshes. Conversely, the Nanite implementation exhibits noticeably sparser coverage. This reduction in visible density correlates with the high fragment processing cost associated with alpha-masked foliage. Because Nanite must evaluate the alpha mask during the visibility pass to determine transparency [2], the engine appears to aggressively cull instances to mitigate the heavy pixel shading workload.

Scene 2 reveals spatial anomalies in foliage instantiation near water boundaries. The LOD pipeline renders grass seamlessly across the terrain margin. In contrast, the Nanite pipeline exhibits barren patches where foliage fails to render. This aggressive culling is likely a consequence of the severe VRAM pressure introduced by Nanite's streaming pool. As memory approaches the 8 GB limit of the test hardware, the engine's streaming manager must aggressively cycle data across the

(a) Scene 1: LOD grass rendering



(b) Scene 1: Nanite grass rendering

(c) Scene 2: LOD terrain with dense grass coverage

(d) Scene 2: Nanite terrain with sparse grass coverage

(e) Scene 3: LOD tree rendering

(f) Scene 3: Nanite tree rendering

Fig. 4: Visual comparison between traditional LOD and Nanite rendering across three representative scenes. Scene 1 highlights differences in grass density. Scene 2 illustrates terrain coverage differences where Nanite exposes more of the underlying water surface. Scene 3 demonstrates rendering artifacts observed in Nanite foliage and shadow handling compared to the stable LOD rendering.

PCIe bus, a known hardware limitation when managing extremely dense virtualized scenes [9]. This results in dropped geometry clusters in dense, dynamic areas.

Furthermore, Scene 3 highlights discrepancies in complex asset shading and shadow evaluation. While the LOD tree maintains consistent self-shadowing and diffuse rendering, the Nanite equivalent displays abnormal, highly specular white shading artifacts along the foliage edges. This visual degradation aligns directly with our GPU render pass profiling (as detailed in Table II and Fig. 6), which recorded a 52.8% increase in shadow depth cost and a 120% increase in deferred lighting execution time. The virtualized geometry pipeline inherently struggles to accurately resolve lighting and shadow bounds on heavily animated, alpha-masked vertex data, as its architecture is fundamentally optimized for opaque, rigid meshes [9].

D. Memory Consumption

The Nanite configuration consumed 7.20 GB of GPU memory, whereas the LOD pipeline used 5.04 GB (+42.9%). This massive increase is due to the VisBuffer allocation and the streaming pool required to keep cluster data resident in memory.

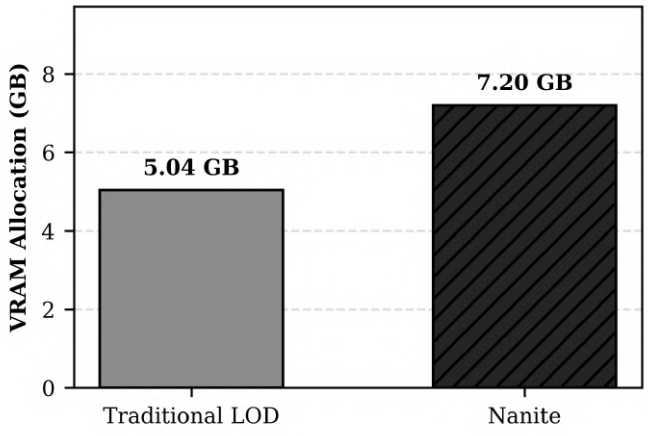


Fig. 5: GPU memory usage comparison between Nanite and traditional LOD.

E. GPU Render Pass Analysis

Profiling individual render passes reveals where Nanite loses time. Nanite introduced the visibility buffer (3.42 ms) and base pass (1.57 ms). Shadow depth cost increased heavily from 3.39 ms to 5.18 ms (+52.8%), and Deferred Lighting increased from 0.75 ms to 1.65 ms (+120%).

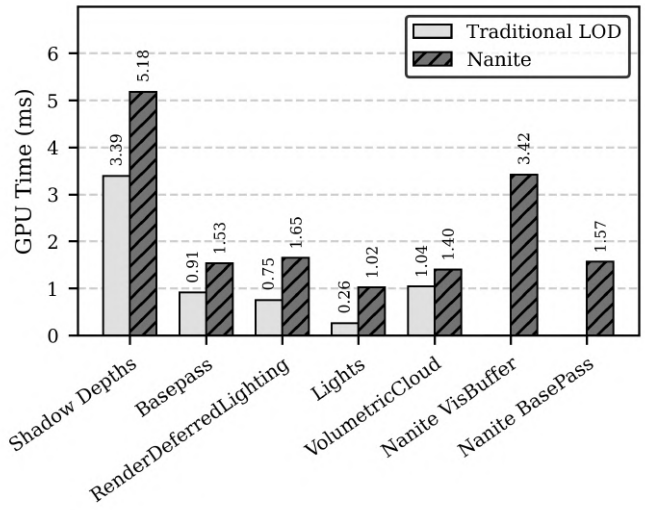


Fig. 6: GPU render pass timing comparison highlighting Nanite visibility buffer cost and increased shadow overhead.

F. Shader Complexity Analysis

Shader complexity visualization reveals a substantial increase in pixel shading cost for the Nanite configuration. The traditional LOD pipeline remains largely in the low-cost region, whereas the Nanite configuration exhibits extensive high-cost areas. This behavior is caused by the increased number of visible micro-triangles and overdraw in alpha-masked foliage, which leads to higher fragment processing cost. The result confirms that the performance degradation is not purely geometric but strongly influenced by pixel shading workload.

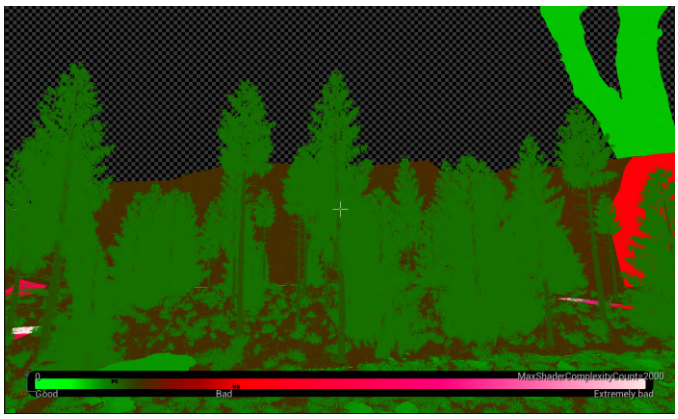


Fig. 7: Shader complexity visualization for the traditional LOD pipeline showing predominantly low-cost (green) foliage shading.

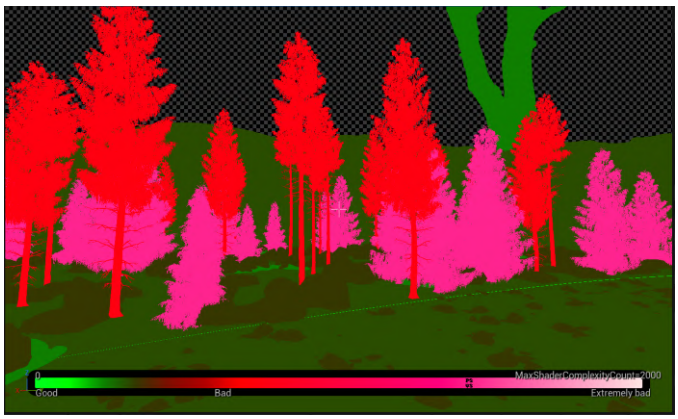


Fig. 8: Shader complexity visualization for the Nanite pipeline showing significantly higher pixel cost (red) in dense foliage regions.

VII. Discussion

A. The WPO Coherence Problem

The test scene is dominated by foliage with wind animation. Nanite's hierarchical culling is fundamentally built on the assumption of rigid geometry. When vertices are continuously deformed via WPO, cluster bounds become invalidated or must be heavily padded. Consequently, the GPU software rasterizer is forced to conservatively process a significantly larger number of clusters, drastically increasing the VisBuffer generation time (3.42 ms). In contrast, traditional LODs process WPO directly in the vertex shader on pre-simplified geometry without needing to evaluate hierarchical cluster visibility.

B. The VRAM Ceiling and Hardware Constraints

The 42.9% increase in VRAM allocation (reaching 7.20 GB) pushes the 8GB limit of the RTX 4060. When VRAM approaches capacity, the Unreal Engine streaming manager must

aggressively cycle data across the PCIe bus, introducing micro-stutters and cache pressure. While Nanite is theoretically capable of rendering infinite detail, on mid-range consumer hardware, the memory bandwidth required to stream and store these massive DAG structures acts as a hard performance ceiling in dense, open-world environments.

C. Overdraw and Masked Materials

Foliage rendering relies heavily on "Masked" opacity materials to simulate leaves. Nanite's primary strength is its VisBuffer, which evaluates materials only once per pixel, eliminating overdraw. However, masked materials require the rasterizer to evaluate the alpha mask *during* the visibility pass to know if a pixel is transparent. The shader complexity results further support the GPU-bound interpretation. The increased pixel cost in the Nanite configuration leads to higher lighting, shadow, and base-pass times, which directly contributes to the observed frame-time increase. This forces Nanite out of its ultra-fast opaque path, negating the benefits of micro-polygon rendering and explaining the massive spike in the draw thread (23.00 ms) as the engine struggles to synchronize masked cluster data.

D. Practical Guidelines for Developers

Based on these findings, we propose a hybrid approach to open-world geometry optimization:

TABLE III: Geometry Optimization Decision Matrix

Asset Type / Scenario	Use Nanite	Use Traditional LOD
Static Architecture (Buildings, Cliffs)	✓	
Hero Assets (Statues, Main Props)	✓	
Dense Animated Foliage (Grass, Trees)		✓
Distant Background Meshes		✓
Hardware target with $\leq$ 8GB VRAM	Use Sparingly	✓

VIII. Limitations

This study is limited to a single hardware configuration (RTX 4060). The impact of different GPU architectures (e.g., AMD RDNA3) and standard Unreal Engine upscaling techniques, such as Deep Learning Super Sampling (DLSS) or Temporal Super Resolution (TSR), were not evaluated. Upscaling could potentially mitigate Nanite's high base-pass cost by rendering the VisBuffer at a much lower internal resolution.

IX. Future Work

Future research will extend this evaluation to include upscaling technologies (DLSS/TSR) to observe if lowering the internal resolution scales linearly with Nanite's VisBuffer overhead. Additionally, the development of dynamic hybrid pipelines—where the engine automatically falls back from Nanite to traditional billboard LODs based on distance and wind intensity—represents a vital area for future rendering optimization.

X. Conclusion

This paper presented a real-time performance trade-off analysis between Nanite and traditional LOD systems in a dynamic open-world environment using Unreal Engine 5.7. The findings demonstrate that while Nanite achieves a remarkable 95.4% reduction in geometric primitives and eliminates CPU draw call bottlenecks, it incurs a 27.6% increase in GPU processing time and a 42.9% higher VRAM footprint in foliage-heavy scenes. The loss of cluster coherence due to vertex animation and the cost of evaluating masked materials render Nanite less efficient than traditional LODs for animated vegetation. Ultimately, developers should adopt a hybrid approach, leveraging Nanite, which is highly efficient for dense *rigid* geometry such as architectural structures and hero assets, where cluster coherence is preserved and pixel overdraw is limited. However, in dense foliage environments with alpha-masked materials and vertex animation, the increased visibility processing and fragment shading cost reduce its effectiveness. In such scenarios, traditional LOD systems provide superior real-time performance. These findings challenge the conventional assumption that triangle count is the dominant performance metric in modern real-time rendering and instead highlight memory bandwidth and visibility processing as the primary limiting factors.

References

[1] W. Shiyun, "Level of detail optimization for real-time terrain," in *2020 International Conference on Computer Engineering and Application (ICCEA)*. IEEE, 2020.

[2] B. Karis, R. Stubbe, and G. Wihlidal, "Nanite: A deep dive," in *SIGGRAPH 2021 Advances in Real-Time Rendering in Games Course*. Epic Games, 2021.

[3] M. Bao, Z. Tao, X. Wang, J. Liu, and Q. Sun, "Comparative performance analysis of rendering optimization methods in unity tuanjie engine, unity global and unreal engine," in *2024 IEEE Smart World Congress (SWC)*. IEEE, 2024.

[4] M. W. Pettett, "Multiresolution mesh rendering engine: Practicalities and performance," University of Cambridge, Department of Computer Science and Technology, Tech. Rep., 2024, supervised by Rafał K. Mantiuk.

[5] Epic Games, *Nanite for Educators and Students*, Epic Games, 2022.

[6] S. R. Overton, "Lods and nanite within unreal engine 5: The future of 3d asset creation for game engines," Master of Fine Arts in Digital Media Culminating Experience, East Tennessee State University, May 2024.

[7] N. Fedotova, M. Protsenko, I. Baranova, S. Vashchenko, and Y. Dehtiarenko, "Research on calculation optimization methods used in computer games development," *IAPGOŚ*, vol. 3, 2023.

[8] O. Sobchyshak, S. Berrezueta-Guzman, and S. Wagner, "Pushing the boundaries of immersion and storytelling: A technical review of unreal engine," *Displays*, vol. 91, p. 103268, 2026.

[9] F. Hussain, "Seminar report on nanite," Islamic University of Science and Technology, Awantipora, Pulwama, J&K, Tech. Rep., Autumn 2024.