

Article

Decoupled Heightfield Tessellation: Eliminating Adjacency Dependency for High-Throughput Spatial Computing

Eun-Seok Lee 

Department of AI Content Engineering, Kangwon National University,
Wonju-si 26403, Gangwon-do, Republic of Korea; elflee77@gmail.com

Abstract

Modern terrain rendering frameworks often struggle with the inherent trade-off between geometric continuity and hardware utilization. State-of-the-art watertight tessellation methods typically rely on explicit adjacency management and multi-pass synchronization, which severely restrict GPU parallelism and induce significant performance spikes during rapid camera transitions. In this paper, we propose a high-throughput, decoupled terrain rendering architecture that ensures visual continuity without the need for horizontal dependency tracking. Our framework leverages a linearized binary tree maintained within a 1D Unordered Access View (UAV), enabling each GPU thread to perform refinement and geometric reconstruction in total isolation. To resolve T-junction artifacts, we introduce an adjacency-blind directional stitching mechanism that utilizes localized hierarchical flags and per-edge error metrics to perform deterministic geometric repair in a single execution pass. Experimental results demonstrate that the proposed method achieves a stable rendering throughput of over 170 FPS on a large-scale 8192×8192 dataset. Notably, our approach provides a 14% improvement in frame-rate stability during instantaneous camera teleportation compared to existing watertight frameworks, while simultaneously reducing the average triangle count by 21% through optimized boundary repair. These results confirm that our decoupled pipeline offers a robust and scalable solution for real-time spatial computing and high-performance simulation environments.

Keywords: terrain rendering; level-of-detail; adjacency-blind stitching; GPU optimization

1. Introduction

Real-time visualization of large-scale terrain data is a critical component in modern interactive graphics applications, including extended reality (XR) content [1], high-fidelity games [2,3], and digital twin frameworks [4]. These systems require a seamless representation of extensive geospatial elevation models to support immersive user experiences. However, the sheer volume of high-resolution heightfield data often exceeds the immediate processing and memory capacities of contemporary hardware, making it impossible to render the entire dataset at full resolution in a single pass.

To address this challenge, view-dependent Level of Detail (LoD) techniques [5–7] are employed to dynamically adjust the geometric complexity of the terrain mesh based on the spatial relationship of camera to the surface. By allocating more polygons to regions near the observer and simplifying distant areas, these methods maintain a balance between visual quality and computational efficiency. Given the intensive nature of per-vertex height sampling and mesh construction, it is imperative that these LoD algorithms are executed directly on the Graphics Processing Unit (GPU) to leverage its massive parallelism.



Academic Editor: Harry Agius

Received: 22 April 2026

Revised: 21 June 2026

Accepted: 26 June 2026

Published: 23 July 2026

Copyright: © 2026 by the author.

Licensee MDPI, Basel, Switzerland.

This article is an open access article distributed under the terms and conditions of the [Creative Commons Attribution \(CC BY\)](https://creativecommons.org/licenses/by/4.0/) license.

The transition to GPU-driven pipelines is primarily motivated by performance constraints, as the overhead of frequent CPU-to-GPU data transfers can severely bottleneck the rendering throughput of large-scale environments [2,8,9]. Consequently, modern terrain engines [10–12] must utilize advanced GPU features, such as compute shaders or hardware tessellation, to achieve the frame rates required for real-time interaction. Ensuring stable and high-throughput performance under varying camera perspectives remains a pivotal issue in the design of scalable terrain rendering systems.

Recent research in terrain rendering has prioritized achieving a mathematically watertight tessellation to eliminate T-junction artifacts. A state-of-the-art approach [13–15] maintains explicit triangle adjacency in GPU memory and performs incremental updates via view-dependent metrics. While this ensures a continuous surface, it introduces a strict adjacency dependency that severely restricts GPU parallelism. To prevent data race conditions during concurrent memory access, these frameworks rely on complex task scheduling, such as graph coloring, which necessitates multiple sequential compute shader dispatches (typically 4 to 8 passes) and explicit synchronization points. Furthermore, these methods often employ recursive tessellation loops to reach a target LoD within a single frame. This leads to unpredictable computational spikes during rapid camera transitions, undermining the frame-rate stability required for high-performance spatial computing. The overhead of re-sampling all cached vertices in every frame, combined with the latency of pointer-heavy index manipulation, creates a significant bottleneck. Consequently, there is a critical need for a decoupled, high-throughput framework that ensures visual continuity through self-contained geometric stitching, bypassing the performance penalties of strict adjacency management.

To address these challenges, we propose a high-throughput terrain rendering framework that achieves visual continuity through a decoupled, adjacency-blind boundary stitching mechanism. Unlike existing methods that enforce watertightness via expensive neighbor-state queries and multi-pass synchronization, our approach leverages a linearized binary tree structure where each node independently manages its topological state using localized 3-bit boundary flags. By pre-calculating per-edge geometric error metrics, the geometry shader can deterministically predict and seal potential T-junction gaps in a single execution pass. This “adjacency-blind” design eliminates horizontal data dependencies, allowing the GPU to process primitives at near-peak occupancy without the latency of graph-based scheduling. Furthermore, we introduce a temporal coherency constraint that limits the refinement level to a single step per frame, effectively flattening computational spikes and ensuring stable frame rates during high-speed camera movement. Experimental results demonstrate that our framework provides a substantial performance gain over current state-of-the-art watertight schemes while maintaining a strictly artifact-free terrain surface.

The remainder of this paper is organized as follows. Section 2 reviews related work in the field of terrain rendering and GPU-based tessellation techniques, focusing on the evolution of adjacency management. Section 3 provides a detailed description of the proposed decoupled tessellation method, including the data structures for dirty flags and the skirt generation logic. In Section 4, we evaluate the performance and accuracy of our method through a series of comparative experiments with state-of-the-art watertight approaches. Finally, Section 5 concludes the paper and discusses potential directions for future research.

2. Related Works

The evolution of large-scale terrain rendering has been driven by the need to manage massive geospatial datasets that exceed standard graphics hardware capabilities. Early re-

search primarily established hierarchical multiresolution models, such as quadtrees [16,17], bintrees [18,19], and Triangulated Irregular Networks (TIN) [20], to implement view-dependent Level of Detail (LoD). Algorithms such as Real-time Optimally Adapting Meshes (ROAM) [18] provided a foundational framework for continuous LoD by dynamically splitting and merging triangles. However, these traditional approaches were heavily reliant on CPU-side computations to maintain mesh topology and ensure geometric continuity. The necessity of constant CPU intervention created a significant communication bottleneck, as updated vertex and index data had to be frequently transferred to the GPU, limiting the overall rendering throughput. To overcome these constraints, subsequent methods like geometry clipmaps and nested regular grids shifted the workload to the GPU by using pre-allocated buffers. While these grid-based approaches offered high performance, they often lacked the fine-grained adaptivity required for complex, non-uniform terrain features.

The introduction of dedicated hardware tessellation units and programmable compute shaders marked a significant shift toward fully GPU-driven terrain rendering pipelines [12,21]. By utilizing the hull and domain shader stages, modern engines can adaptively subdivide patches in real time based on distance-based or screen-space error metrics [11]. These hardware-accelerated methods significantly reduce the CPU load by generating geometric detail directly on the graphics card. However, traditional hardware tessellation is inherently stateless, meaning the entire mesh must be reconstructed from scratch in every frame regardless of camera movement [13]. This lack of temporal data reuse leads to redundant vertex sampling and interpolation, especially when high-order surface reconstruction is required for scientific or high-fidelity visualization [22]. Furthermore, most hardware-based approaches are constrained by a maximum edge tessellation factor (typically 64), which limits the achievable level of detail in extreme close-up views [23]. These limitations have prompted the development of more flexible, compute shader-based frameworks that can maintain and incrementally update the triangulation state across successive frames [13,14].

Recent advancements in compute shader-based terrain rendering have introduced incremental tessellation techniques that cache the triangulation of the previous frame to minimize redundant computations [14]. By subdividing or merging only the necessary triangles, these methods achieve high performance even with complex, higher-order surface reconstruction [22]. However, a significant challenge in these parallel frameworks is maintaining a watertight surface without visual artifacts such as T-junctions [23]. To ensure geometric continuity, state-of-the-art approaches maintain explicit adjacency and hierarchy information for each triangle in GPU memory [13,14]. While effective, this imposes a strict adjacency dependency where updating a single triangle requires synchronized modifications to its neighbors. To avoid data race conditions, these systems often employ task scheduling via graph coloring, which necessitates multiple sequential compute shader dispatches and explicit synchronization points per frame [13]. This multi-pass requirement (typically four to eight passes) substantially increases implementation complexity and limits the peak throughput of the GPU. Furthermore, recursive tessellation loops designed to reach a target LoD within a single frame can lead to unpredictable computational spikes, undermining the stability required for high-performance spatial computing [13,14].

3. Dual-Phase Terrain Generation with Temporal Coherency and Adjacency-Blind Boundary Stitching

The primary objective of the proposed method is to establish a high-throughput terrain rendering pipeline by decoupling the logical refinement of the Level of Detail (LoD) hierarchy from the physical generation of the mesh. Traditional watertight frameworks often suffer from significant synchronization overhead due to their reliance on explicit adja-

gency tracking and multi-pass graph coloring to resolve T-junctions. These conventional approaches create horizontal dependencies where the subdivision state of a node is contingent upon its neighbors, thereby restricting the independent execution of GPU threads and inducing complex memory access latencies. To eliminate these constraints, this research introduces a two-phase execution model that processes each primitive in total isolation while maintaining a perfectly watertight surface through localized boundary repair.

The overall pipeline is structured into a detail level search stage (Phase 1) and a mesh generation stage (Phase 2). In Phase 1, the system traverses a linearized binary tree buffer and updates the state of each node based on a view-dependent error metric, recording the results via persistent dirty flags. The indices of the updated nodes are then streamlined through the hardware-accelerated stream output stage, passing them as pointers to the subsequent phase. In Phase 2, the geometry shader interprets these pointers to assemble the final triangulation. Instead of querying neighbor states, the shader utilizes internal hierarchical flags and per-edge error metrics to execute selective directional stitching. This design ensures visual continuity at T-junctions through self-contained geometric repair, enabling the completion of the rendering process within a single pass without global synchronization.

This chapter defines the core linearized binary tree data structure and the error metrics utilized in the pipeline. In addition, it details the logic behind the incremental refinement mode and the immediate reconstruction mode, demonstrating how the system maintains stability during continuous camera movement and rapid transitions. Specifically, the following sections present the detailed algorithm for performing adjacency-blind stitching within the geometry shader to optimize the rendering workload by avoiding redundant primitive assembly while ensuring a strictly watertight representation.

The pipeline, as illustrated in Figure 1, is divided into two asynchronous stages: Phase 1 (Detail Level Search) and Phase 2 (Mesh Generation). Phase 1 traverses the linearized binary tree to evaluate the LoD metrics and marks dirty flags for topology updates. These updates are emitted via the stream output stage as node pointers. Phase 2 interprets the streamlined node pointers within the geometry shader to reconstruct the terrain's geometric primitives. For each active node, the shader regenerates the surface triangles by decoding its hierarchical level and spatial position from the linearized binary tree buffer. Similarly, the shader executes adjacency-blind boundary stitching by referencing the subdivision flags and metadata stored in the tree. Instead of performing expensive neighbor lookups, the geometry shader utilizes these localized flags to identify boundary edges that are finer than their counterparts in the logical hierarchy. Upon detecting such a disparity, the shader performs an immediate directional subdivision at the boundary to seal potential T-junctions. This process achieves high throughput by ensuring that each primitive is processed in total isolation, using fixed-function hardware for final assembly, while maintaining a robust and strictly watertight terrain surface through flag-driven geometric repair.

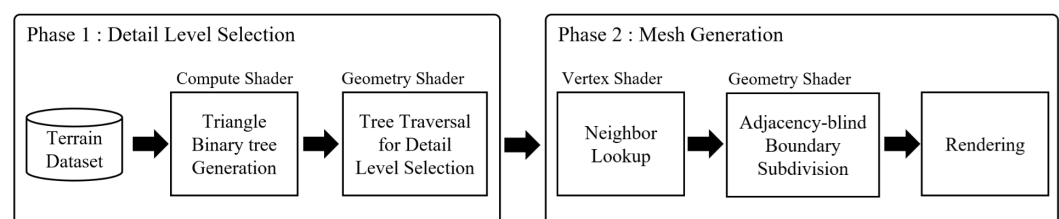


Figure 1. Overview of the proposed dual-phase terrain rendering pipeline highlighting the adjacency-blind boundary stitching process.

3.1. Data Structure and LoD Metric for Decoupled Refinement

The efficiency of the proposed terrain rendering pipeline is fundamentally rooted in a data structure that replaces explicit adjacency searches with arithmetic indexing and deterministic bit-field flags. We represent the terrain hierarchy using a linearized binary tree maintained within a one-dimensional buffer. The linearized binary tree is implemented as a Structured Buffer bound to an Unordered Access View (UAV), enabling each GPU thread to perform high-speed, non-blocking read and write operations. Since our framework is designed for total isolation, each thread in Phase 1 is assigned a unique node index calculated via constant-time arithmetic, ensuring that no two threads attempt to modify the same bit-field simultaneously. This “one-thread-per-node” mapping eliminates the need for expensive atomic operations or global synchronization barriers. The UAV-based architecture facilitates a persistent state management system where the results of the LoD evaluation—specifically the split/merge decisions and the associated subdivision flags—are recorded directly into the 1D tree buffer. This persistent metadata serves as the ground truth for the asynchronous Phase 2, allowing the geometry shader to fetch the refined topological state without re-evaluating the view-dependent metrics.

Unlike conventional quadtree or triangle-bintree implementations that rely on pointer-based linked lists or complex index encoding (which frequently trigger cache misses and stall GPU warp execution) our linearized structure allows for the direct calculation of parent-child and sibling relationships through constant-time arithmetic operations. Each node N_i in this buffer functions as an independent processing unit, storing its hierarchical level L , object-space geometric errors ρ , and a dedicated bit-field for state management, including a 1-bit subdivision flag that indicates whether the node’s children are currently active. This flat memory layout is specifically optimized for high-bandwidth coalesced memory access during the parallel hierarchy refinement in Phase 1.

The metadata associated with each node is meticulously encoded to support adjacency-blind refinement. In addition to pre-calculated per-edge error metrics (ρ_{left} , ρ_{right} , and $\rho_{hypotenuse}$), we introduce a 3-bit boundary flag B_{edge} that encodes the topological context of the triangle within the bisection hierarchy. The B_{edge} flag is encoded using three bits because each node in the longest-edge bisection hierarchy is represented as a triangle. The three bits correspond directly to the left edge, right edge, and hypotenuse edge, respectively. A bit value of 1 indicates that the corresponding edge is inherited from an ancestral hierarchy boundary and may require boundary stitching during mesh reconstruction. This flag identifies which edges correspond to the ancestral boundaries inherited from the parent node. By embedding this structural information directly into the node metadata, the geometry shader in Phase 2 can determine the potential for T-junction formation without performing a single query for neighbor nodes. Specifically, the shader determines whether boundary stitching is required for an edge E using the local subdivision flag and the associated $B_{edge}(E)$ metadata. Since B_{edge} encodes the hierarchical boundary classification of each edge, all the information required for the stitching is available within the current node. Consequently, boundary stitching can be performed without explicit neighbor-state queries, eliminating the strict adjacency dependency found in conventional watertight tessellation frameworks.

From a shader execution perspective, this data-centric design provides a significant advantage in terms of register pressure and instruction throughput. During Phase 1, each thread determines the split or merge state of its assigned node using only local bit-fields and a global LoD map, requiring no inter-thread communication or global locks. The resulting state is recorded as a persistent dirty flag, which serves as the primary control signal for Phase 2. In Phase 2, the geometry shader interprets the streamlined node pointers to reconstruct the surface triangles by decoding the hierarchical level and spatial position directly

from the 1D buffer. By offloading the topological decision-making to these lightweight bit-fields, the shader bypasses the complex recursive logic typically required to enforce watertightness. Instead, it performs a deterministic bisection and directional stitching based on the local flags, maintaining near-peak occupancy and providing headroom for high-order interpolation.

The refinement of the terrain hierarchy is dynamically governed by a view-dependent LoD decision metric. To ensure high-throughput processing, the system calculates a screen-space error threshold T for every active node during Phase 1:

$$T = \frac{k \cdot \rho_{max}}{d \cdot \cos(\theta)} \quad (1)$$

where ρ_{max} denotes the maximum object-space geometric error among the triangle's three edges, k is a user-defined quality factor, d is the Euclidean distance to the node center, and θ is the angle between the view vector and the surface normal. The calculated T is then compared against predefined thresholds, ϵ_{split} and ϵ_{merge} , to trigger independent state transitions. This localized evaluation ensures a completely decoupled refinement process.

The depth of the directional stitching, selectively generated in Phase 2, is further optimized through the integration of per-edge error metrics. When the geometry shader identifies an edge as a potential T-junction based on the subdivision and B_{edge} flags, it calculates the required extrusion depth H using the corresponding edge-specific error (ρ_{left} , ρ_{right} , or $\rho_{hypotenuse}$):

$$H = \alpha \cdot \rho_{edge} + \beta \quad (2)$$

where α and β are scaling factors calibrated to the heightmap resolution. This adaptive approach ensures that the vertical repair is sufficiently deep to seal floating gaps or submerged overlaps without introducing excessive geometric redundancy. This synergy between the 1D tree data structure and the adaptive stitching logic facilitates a hardware-friendly pipeline where the fixed-function primitive assembly units operate at maximum efficiency, proving the practical superiority of our decoupled framework in managing large-scale, dynamic terrain environments.

The Phase 1 execution procedure is straightforward. Each compute shader thread is assigned a unique node index from the linearized binary tree. The thread first evaluates the screen-space error metric T and compares it against the predefined split and merge thresholds. Based on this evaluation, the corresponding subdivision flag and dirty state are updated within the UAV buffer. Finally, the indices of active nodes are emitted through the stream output stage and forwarded to Phase 2 for mesh generation.

3.2. Incremental Refinement and Geometry Shader-Based Mesh Generation

The proposed framework achieves stable rendering performance and visual continuity by employing an incremental refinement strategy that limits the level-of-detail transition to a maximum of one level per frame. During Phase 1, the processing sequence consists of four steps: (1) traversal of the active nodes in the linearized binary tree, (2) evaluation of the view-dependent LoD metric, (3) update of subdivision and dirty flags, and (4) emission of active node indices through stream output. Since each node is processed independently, these operations can be executed concurrently without neighbor synchronization or recursive hierarchy traversal.

This temporal coherency constraint is integrated into the Phase 1 traversal, where each thread in the compute shader independently evaluates its assigned node in the UAV-bound linearized binary tree. By restricting the subdivision or merging to a single hierarchical step, the system prevents sudden computational spikes that typically occur in recursive tessellation loops. The resulting state transitions are recorded directly into

the UAV as persistent subdivision flags and dirty bits. Once the topology is finalized, the indices of all active leaf nodes are emitted via the *Stream Output* stage, allowing Phase 2 to focus exclusively on the primitives that constitute the current physical mesh. Phase 1 is implemented as a compute shader pass operating on the linearized binary tree stored in the UAV buffer. The compute shader is executed using a fixed thread-group configuration (in our experimental case, we used 128,1,1), where each thread is assigned to a single node and performs an independent LoD evaluation. This one-thread-per-node mapping eliminates inter-thread dependencies and allows the hierarchy update stage to scale efficiently with scene complexity.

The core innovation of the mesh generation phase lies in its ability to resolve T-junctions through adjacency-blind directional stitching. Phase 2 proceeds in three stages. First, the geometry shader retrieves the node metadata associated with each emitted pointer. Second, the corresponding terrain primitive is reconstructed using the hierarchical level and spatial coordinates stored in the linearized buffer. Third, boundary stitching is evaluated using the subdivision flags and Bedge metadata, and additional stitching geometry is generated when required. Upon receiving the node pointers from Phase 1, the geometry shader reconstructs the surface triangles by decoding the hierarchical level and spatial position from the linearized buffer. The primary challenge in decoupled LEB-based tessellation is the emergence of T-junctions at the boundaries between nodes of differing LoD levels, as analyzed in Figure 2. Instead of resolving these via expensive neighbor-state queries, our strategy utilizes the 3-bit boundary flags B_{edge} and the local 1-bit subdivision flag stored in the UAV. By comparing these localized bit-fields, the geometry shader identifies edges that are more refined than their counterparts in the logical hierarchy in constant time.

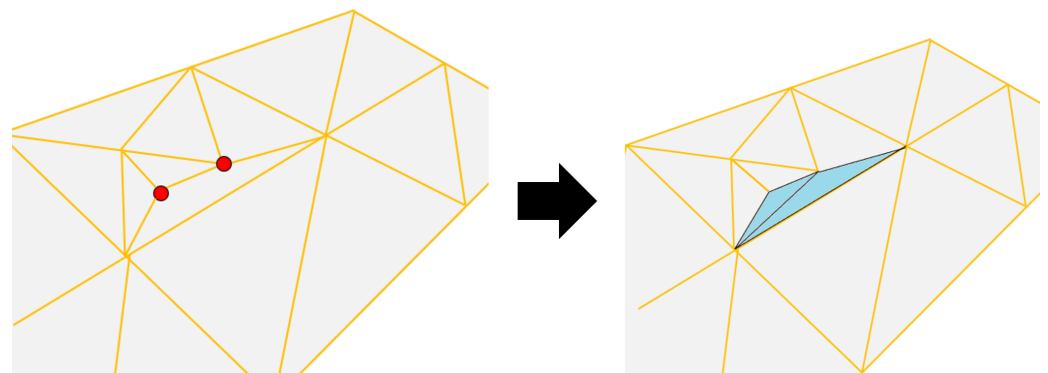


Figure 2. T-junction method for our approach. The red vertex is T-vertex, and the cyan triangles are the result of performing T-junctions.

For every edge identified as a T-junction candidate, the geometry shader executes a directional stitching process to ensure a strictly watertight surface. Instead of generating redundant global geometry, the shader performs a localized geometric repair by extruding a vertical boundary mesh. Each geometry shader invocation operates on a single active primitive and evaluates boundary stitching independently. When stitching is required, only a limited number of additional stitching primitives are generated along the affected boundary edges. Consequently, the geometry amplification remains bounded and does not introduce excessive primitive expansion during mesh reconstruction. The depth of this stitching, denoted as H , is dynamically adjusted based on the per-edge error metrics ρ_{edge} stored in the tree buffer. By scaling the repair depth as a function of the local terrain roughness, the system ensures that the vertical seal is sufficient to cover any floating gaps or submerged overlaps, regardless of the topographic complexity.

This hardware-driven approach leverages fixed-function units for high-throughput primitive assembly while the programmable stages focus on deterministic, flag-driven geometric repair. The resulting mesh maintains visual continuity even during rapid camera movements, proving the practical superiority of our decoupled framework in managing massive-scale terrain environments. By bypassing the performance penalties of strict adjacency management and multi-pass synchronization, the pipeline maintains near-peak occupancy and provides a robust solution for real-time spatial computing.

3.3. Immediate Reconstruction and Hardware-Accelerated Geometric Stitching

While the incremental refinement mode ensures temporal stability, the immediate reconstruction mode provides the necessary flexibility for global transitions, such as camera teleportation, by bypassing the one-level-per-frame limitation. The technical challenge in such rapid transitions is maintaining visual continuity when multiple hierarchy levels are subdivided or merged simultaneously. Our framework resolves this by ensuring that the Phase 2 mesh generation logic remains identical across both modes, leveraging the geometry shader to perform real-time primitive expansion from the node pointers updated in Phase 1. Each input vertex to the geometry shader carries a 32-bit encoded index that functions as a pointer to the 1D tree buffer, allowing the shader to fetch the most recent refinement flags and boundary metadata even after a massive structural update in the UAV.

The core of this high-speed reconstruction lies in the geometry shader's ability to interpret the 3-bit boundary flags B_{edge} and execute directional stitching in constant time. As the hierarchy is reconstructed, the shader determines the necessity of boundary stitching using the current node's subdivision state and the precomputed boundary descriptors stored in the linearized tree. Because all stitching decisions are derived from node-local metadata, immediate reconstruction can be performed without neighbor-state queries or adjacency synchronization. Because the 1D tree buffer provides a synchronized view of the entire hierarchy, the shader can immediately detect whether a boundary requires a vertical repair to seal geometric disparities. In the case of a Floating Gap resulting from a sudden multi-level subdivision, the shader calculates the elevation difference relative to the coarse neighbor's edge and extrudes a fitting geometric patch. This directional logic ensures that even when the terrain topology changes drastically in a single frame, the mesh remains visually watertight without requiring the cascading neighbor updates or recursive synchronization passes found in conventional frameworks.

The immediate reconstruction mode follows the same processing sequence as Phase 2. The only difference is that multiple hierarchy levels may be updated simultaneously in Phase 1. Once the updated node pointers are generated, mesh reconstruction and boundary stitching are performed using the same node-local metadata without additional synchronization stages.

The efficiency of this mode is further amplified by the seamless handling of node merging within the geometry shader. During a global transition where multiple sub-triangles are consolidated, Phase 1 updates the hierarchical flags in the 1D UAV to reflect the inactive state of child nodes. The geometry shader detects this flag convergence and instantly ceases the emission of refined sub-triangles, outputting the consolidated parent primitive instead. This deterministic mapping between the tree pointers and the emitted geometry allows the immediate reconstruction mode to achieve a complete LoD update within a single high-throughput compute dispatch. By eliminating the need for adjacency-forced subdivisions or complex task scheduling such as graph coloring, the system maintains a high degree of SIMD occupancy. This ensures that the physical mesh is always a direct and watertight reflection of the logical hierarchy, regardless of the scale or speed of the topological changes, effectively providing a self-contained geometric repair mechanism.

4. Experimental Results and Discussion

The performance of the proposed framework was evaluated on a workstation equipped with an Intel Core i5-13600K CPU and an NVIDIA GeForce RTX 3060 GPU (12 GB VRAM). The system was implemented using the Unity3D engine (Version 2022.3 LTS) targeting the DirectX 12 API on a Windows 11 environment. To maintain consistent visual fidelity and focus on geometric performance, we applied Unity's standard Lambertian shading model with a high-resolution normal map. To ensure a rigorous and fair comparison, all experiments were conducted under a strict screen-space error tolerance of less than 0.5 pixels, ensuring that no visual artifacts or T-junctions were perceptible. We compared our method against the state-of-the-art watertight framework presented in IEEE TVCG [13] and traditional quadtree/bintree-based CLOD methods. The evaluation was performed using two distinct high-resolution datasets [24]: a custom Digital Elevation Model (DEM) of Wonju-si, South Korea (8192×8192), which features steep and complex mountainous terrain.

To analyze the impact of geometric load, we categorized the camera perspectives into Near-view (ground-level exploration) and Far-view (aerial surveillance). In the Near-view scenario, the geometric error ρ increases significantly due to the proximity to the terrain, leading to a massive expansion of the LoD hierarchy. To ensure a fair comparison, occlusion culling was disabled for all evaluated methods. This configuration allows the experiments to measure the raw tessellation and mesh-generation performance without visibility-dependent performance variations. Figure 3 provides visual evidence of the rendering quality and watertight continuity achieved by our framework across both datasets and views.

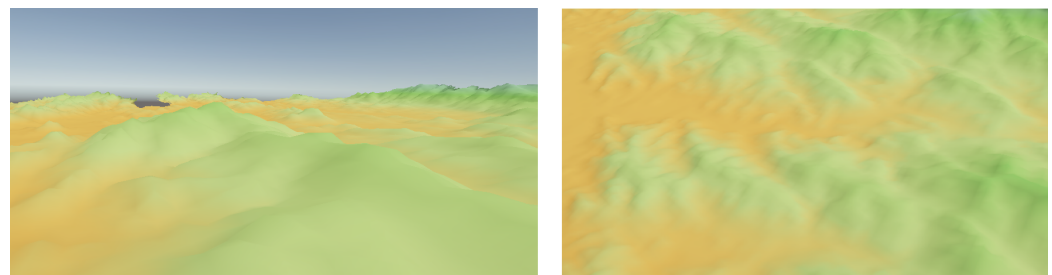


Figure 3. Rendering results under 0.5 pixel error constraint. Left scene is Far-view, and right scene is Near-view of the Wonju-si dataset.

The quantitative data presented in Table 1 demonstrates that the proposed framework significantly outperforms the state-of-the-art watertight scheme in both rendering throughput and structural stability. A critical observation is the minimum FPS recorded during instantaneous camera teleportation; while the current method [13] suffers from a severe performance drop down to 42.5 FPS due to cascading node updates and synchronization bottlenecks, our approach maintains a stable 128.6 FPS. This represents a 202.5% improvement in worst-case stability, confirming that the decoupled, adjacency-blind reconstruction mechanism effectively flattens computational spikes during drastic LoD transitions.

Furthermore, the data reveals a substantial reduction in geometric redundancy. As shown in the table, our method reduces the average triangle count by approximately 21% (from 218,420 to 172,500) while maintaining a strictly watertight surface. This optimization directly translates into a higher average frame rate of 172.4 FPS. Unlike conventional adjacency-based frameworks that often trigger forced subdivisions of neighboring nodes to resolve T-junctions, our directional stitching mechanism performs localized geometric repair only where necessary. By eliminating the multi-pass synchronization overhead and reducing memory controller pressure through linearized arithmetic indexing, the pro-

posed pipeline significantly enhances computational throughput and minimizes hardware idling. Consequently, these results prove that our framework provides a high-throughput architecture specifically optimized for large-scale, dynamic terrain environments.

Table 1. Quantitative performance comparison between the state-of-the-art watertight framework [13] and the proposed method under a 0.5-pixel error tolerance.

Metric	Cornel et al. [13]	Proposed Method	Improvement
Avg. FPS (Overall)	151.2	172.4	+14.0%
Min. FPS (Teleportation)	42.5	128.6	+202.5%
Avg. Triangle Count	218,420	172,500	−21.0%
Peak Frame Time (ms)	23.5	7.8	−66.8%
GPU Memory Usage (MB)	412	325	−21.1%

Although the proposed framework successfully eliminates adjacency synchronization overhead and improves execution stability, several architectural considerations should be acknowledged. The current implementation relies on a conventional tessellation and Geometry Shader pipeline for localized boundary stitching operations. While the amount of primitive amplification remains relatively limited, Geometry Shader throughput may become a scalability constraint under extremely dense tessellation workloads.

Furthermore, the proposed framework has been evaluated using a single GPU configuration and a representative large-scale terrain dataset. Although the underlying linearized 1D terrain representation and localized boundary-processing strategy are largely independent of a specific graphics pipeline stage, their behavior under alternative rendering architectures has not yet been investigated. In particular, adaptation to modern mesh/task shader pipelines and massively parallel GPU architectures remains an important direction for future research.

5. Conclusions

In this research, we presented a decoupled terrain rendering architecture that addresses the scalability limitations of conventional adjacency-managed frameworks. By replacing multi-pass synchronization and cascading refinement with an adjacency-blind boundary stitching mechanism, the proposed method enables a more independent execution model for GPU-based terrain processing. Experimental evaluations on an NVIDIA RTX 3060 demonstrated a 14.0% increase in average rendering throughput and a 21.0% reduction in geometric redundancy compared to a state-of-the-art watertight terrain rendering scheme. Furthermore, the proposed framework achieved a 202.5% improvement in frame-rate stability during rapid camera transitions, significantly reducing the computational spikes commonly observed in dependency-driven terrain rendering pipelines.

These results indicate that eliminating horizontal dependencies through a linearized one-dimensional terrain representation can provide an effective approach for improving both rendering efficiency and execution stability in large-scale terrain environments. The proposed framework demonstrates that adjacency-independent terrain processing can serve as a practical alternative to conventional synchronization-intensive approaches while maintaining visual continuity across terrain boundaries.

Although the proposed framework demonstrated favorable performance improvements, several limitations remain. The current evaluation was conducted using a single GPU configuration and a limited set of terrain datasets, and therefore may not fully represent performance characteristics across diverse hardware platforms and rendering scenarios. Furthermore, the present study focuses primarily on application-level performance metrics, including rendering throughput, frame-time behavior, and memory consumption. While

low-level GPU profiling metrics such as shader occupancy, memory throughput, cache utilization, and synchronization behavior could provide additional insight into the execution characteristics of the proposed framework, such analyses fall beyond the scope of the current work.

Funding: This research was supported by Korea Institute for Advancement of Technology (KIAT) grant funded by the Korea Government (MOTIE) (P0011930, The Establishment Project of Industry-University Fusion District).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The original data presented in the study are openly available from National Geographic Information Institute at <https://map.ngii.go.kr>, accessed on 19 April 2026.

Conflicts of Interest: The author declares no conflicts of interest.

References

1. Lee, Y.; Moon, C.; Ko, H.; Lee, S.H.; Yoo, B. Unified representation for XR content and its rendering method. In Proceedings of the 25th International Conference on 3D Web Technology, Virtual Event, Korea, 5–13 November 2020; pp. 1–10.
2. Li, R. Real-world large-scale terrain model reconstruction and real-time rendering. In Proceedings of the 28th International ACM Conference on 3D Web Technology, San Sebastian, Spain, 9–11 October 2023; pp. 1–10.
3. Masood, Z.; Jiangbin, Z.; Irfan, M.; Ahmad, I. High-performance virtual globe GPU terrain rendering using game engine. *Comput. Animat. Virtual Worlds* **2023**, *34*, e2108.
4. Wang, P.; Yang, M.; Zhu, J.; Peng, Y.; Li, G. Digital twin-enabled online battlefield learning with random finite sets. *Comput. Intell. Neurosci.* **2021**, *2021*, 5582241. [[CrossRef](#)] [[PubMed](#)]
5. Losasso, F.; Hoppe, H. Geometry clipmaps: Terrain rendering using nested regular grids. In *ACM SIGGRAPH 2004 Papers*; ACM: New York, NY, USA, 2004; pp. 769–776.
6. Xia, Y.; Chen, Y.; Wang, D. Real-Time LOD Rendering of Tire Tracks in Dynamic Terrain. In Proceedings of the 3rd International Conference on Electronic Information Technology and Computer Engineering (EITCE), Xiamen, China, 18–20 October 2019; pp. 206–209.
7. De Berg, M.; Dobrindt, K. On levels of detail in terrains. In Proceedings of the Eleventh Annual Symposium on Computational Geometry, Vancouver, BC, Canada, 5–7 June 1995; pp. 426–427.
8. Yuan, M.; Wang, L.; Ai, B. Parallel terrain node processing and rendering: A GPU-accelerated approach. In *Proceedings of the International Conference on Remote Sensing, Mapping, and Image Processing (RSMIP 2025)*, Sanya, China, 17–19 January 2025; SPIE: Bellingham, WA, USA, 2025; Volume 13650, pp. 141–152.
9. Fu, H.; Yang, H.; Chen, C. Large-scale terrain-adaptive LOD control based on GPU tessellation. *Alex. Eng. J.* **2021**, *60*, 2865–2874. [[CrossRef](#)]
10. Cozzi, P.; Ring, K. *3D Engine Design for Virtual Globes*; AK Peters/CRC Press: Natick, MA, USA, 2011; pp. 1–518.
11. Tatarchuk, N.; Barczak, J.; Bilodeau, B. Programming for real-time tessellation on GPU. *AMD Whitepaper* **2010**, *5*, 1–18.
12. Cantlay, I. DirectX 11 terrain tessellation. *NVIDIA Whitepaper* **2011**, *8*, 3.
13. Cornel, D.; Zechmeister, S.; Gröller, E.; Waser, J. Watertight incremental heightfield tessellation. *IEEE Trans. Vis. Comput. Graph.* **2022**, *29*, 3888–3899. [[CrossRef](#)]
14. Takayama, K. Compatible intrinsic triangulations. *ACM Trans. Graph.* **2022**, *41*, 1–12. [[CrossRef](#)]
15. Hedberg, M. Rendering of Large Scale Continuous Terrain Using Mesh Shading Pipeline. Master's Thesis, Umeå University, Umeå, Sweden, 2022.
16. Pajarola, R. Large scale terrain visualization using the restricted quadtree triangulation. In Proceedings of the IEEE Visualization '98, Research Triangle Park, NC, USA, 18–23 October 1998; pp. 19–26.
17. Pajarola, R.; Antonijuan, M.; Lario, R. Quadtin: Quadtree based triangulated irregular networks. In Proceedings of the IEEE Visualization 2002 (VIS 2002), Boston, MA, USA, 27 October–1 November 2002; pp. 395–402.
18. Duchaineau, M.; Wolinsky, M.; Sigeti, D.E.; Miller, M.C.; Aldrich, C.; Mineev-Weinstein, M.B. ROAMing terrain: Real-time optimally adapting meshes. In Proceedings of the IEEE Visualization '97, Phoenix, AZ, USA, 19–24 October 1997; pp. 81–88.
19. White, M. Real-Time Optimally Adapting Meshes: Terrain Visualization in Games. *Int. J. Comput. Games Technol.* **2008**, *2008*, 753584. [[CrossRef](#)]

20. Li, L.; Kuai, X. An efficient dichotomizing interpolation algorithm for the refinement of TIN-based terrain surface from contour maps. *Comput. Geosci.* **2014**, *72*, 105–121. [[CrossRef](#)]
21. Nießner, M.; Loop, C.; Meyer, M.; DeRose, T. Feature-adaptive GPU rendering of Catmull-Clark subdivision surfaces. *ACM Trans. Graph.* **2012**, *31*, 1–11. [[CrossRef](#)]
22. Patney, A.; Owens, J.D. Real-time Reyes-style adaptive surface subdivision. In *ACM SIGGRAPH Asia 2008 Papers*; Association for Computing Machinery: New York, NY, USA, 2008; pp. 1–8.
23. Mikhaylyuk, M.V.; Timokhin, P.Y.; Maltsev, A.V. A method of Earth terrain tessellation on the GPU for space simulators. *Program. Comput. Softw.* **2017**, *43*, 243–249. [[CrossRef](#)]
24. National Geographic Information Institute. Digital Elevation Model (DEM) Open Data. Available online: <https://map.ngii.go.kr> (accessed on 19 April 2026).

Disclaimer/Publisher’s Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.