How Eigenvalues and Gaussian Methods Intersect in making Anime and VTuber Multi-Perspective Production

Rhio Bimo Prakoso S - 13523123[1]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]rhiobimoprakoso.s@gmail.com, 13523123@std.stei.itb.ac.id

***Abstract*—Tree-based data structures, including Octrees, BRLO-Trees, and Bounding Volume Hierarchies (BVH), have emerged as crucial tools for visual complexity management in the production of anime and VTuber content. By organizing 3D spatial data into hierarchical frameworks, they facilitate real-time rendering, support multi-perspective scene organization, and allow for dynamic object management. Concurrently, mathematical tools such as eigenvalue decomposition and Gaussian sampling refine transformations, rotational calculations, and selective rendering strategies, further enhancing computational efficiency and boosting the quality of final images. This paper offers an expansive analysis of these solutions, presenting how Octree-GS, BRLO-Tree, and other hierarchical structures tackle LOD (Level-of-Detail) demands in scenarios ranging from high-action anime sequences to large-scale, interactive VTuber environments. A detailed methodology for assessing these architectures' efficacy, present experimental findings, and discuss broader implications for the future of anime, gaming, and extended reality platforms. Results suggest that the synergy between tree-based data structures and advanced mathematical methods not only optimizes performance but also contributes substantially to creative flexibility and user immersion in 3D productions.**

***Keywords*—3D rendering, anime production, eigenvalues, Gaussian methods, tree-based data structures, VTuber workflows, LOD strategies**

I. Introduction

The production pipelines for anime and VTuber content have expanded in complexity over the last decade, driven by audience expectations for ever-more immersive and visually arresting experiences. Traditional 2D anime remains beloved worldwide, yet the increasing incorporation of 3D elements—whether for backgrounds, character action sequences, or special effects—has created a need for more efficient data structures and rendering methods. Simultaneously, VTubers (Virtual YouTubers), who animate online personalities through real-time motion capture and rendering, have skyrocketed in popularity, requiring advanced platforms capable of facilitating interactive, real-time 3D streaming for global audiences.

Modern anime production often involves multi-perspective storytelling, where a single sequence may integrate several complex shots: extreme close-ups of characters, sweeping environment pans, and rapid camera transformations to maintain dramatic momentum. Each shot demands careful frame composition, real-time lighting adjustments, and collision detection to ensure narrative cohesion. Meanwhile, the VTuber ecosystem merges real-time performance with audience engagement. VTuber personalities dynamically interact with their environments, switching avatar costumes, manipulating objects, and responding to audience-driven events. This environment must be updated swiftly, with minimal latency, to maintain viewers' sense of immersion.

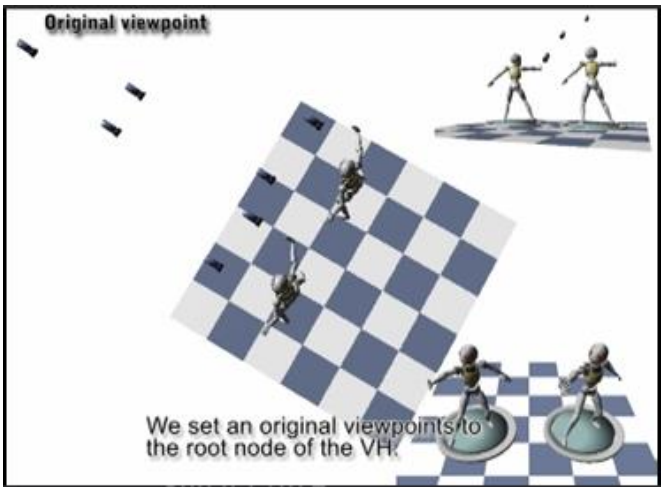

Fig. 1. Example of multi-perspective rendering [1]. The figure shows that by using multiple cameras, which each camera is assigned to a specified body-part/angle, it can create a unique perspective symbolizing that of an authentic anime illustrated scene.

Tree-based data structures have risen as prime candidates to streamline the vast amounts of data processed during rendering and to address multi-view or dynamic scene demands. Octrees subdivide three-dimensional space into cubic nodes, allowing for rapid visibility and collision checks. BRLO-Trees (Block-R-Tree-Loose-Octree hybrids) pool the strengths of different partitioning and bounding strategies to facilitate highly dynamic scenes. Bounding Volume Hierarchies (BVH) have evolved to serve as integral tools in accelerating tasks such as ray tracing and collision detection, making them indispensable in both offline and real-time rendering applications.

Simultaneously, mathematical developments catalyze rendering efficiency where eigenvalues help compress transformation operations by identifying principal axes. Furthermore, Gaussian sampling can concentrate rendering resources in areas of maximum visual interest, enhancing scene fidelity where it matters most while sparing compute power for less critical spaces.

Despite these advancements, adopting such solutions in real-

world production poses multiple challenges. Rendering studios and VTuber production teams often must retrofit existing pipelines to incorporate hierarchical data structures, updating asset management systems, scene authoring tools, and animation workflows. Moreover, the integration of neural rendering techniques—like Neural Radiance Fields (NeRF) or generative adversarial networks (GANs)—adds a new layer of complexity. These emerging methods promise significant leaps in visual fidelity but require balanced synergy with established tree-based frameworks to maintain consistency and real-time adaptability.

Fig. 2. VTuber actively using real-time 3D models tracking for livestreaming. Using interactive element (the online chat) to hold an intriguing program. Source: https://www.youtube.com/live/F7ohLhPMO1s?si=UReX5lN3aN4LIsTx

In what follows, this paper explores the foundations, implementations, and results of integrating tree-based data structures and mathematical optimizations in anime and VTuber productions. A methodology will be outlined for evaluating these solutions, offer both quantitative and qualitative performance analyses, and conclude by discussing the future directions and broader impact of these evolving technologies. By the end, it should be evident that the synergy between hierarchical data structures and advanced mathematical techniques forms not only a key pillar in today's 3D productions but an essential stepping stone toward even more groundbreaking innovations in entertainment and beyond.

II. Background

A. Evolution of Tree-Based Data Structures

1. Historical Roots in Computer Graphics
   Tree-based data structures—particularly Octrees—emerged in computer graphics decades ago, primarily to address the complexities of ray tracing and object culling in scenes with thousands of polygons. Early hardware constraints forced developers to find methods of partitioning space so that certain calculations, like visibility checks, could be avoided when entire regions of space were irrelevant to a particular frame. Over time, as polygon counts ballooned into the millions and eventually billions, these partitioning strategies became indispensable.

2. Adoption in GIS and Gaming
   In Geographic Information Systems (GIS), spatial indexing is paramount for tasks like terrain modeling and city-scale environmental mapping. R-Trees and variants like Quadtree and K-d Trees were used extensively for 2D and 3D geospatial data, laying the foundation for their later appropriation in gaming engines such as Unity and Unreal. These engines, historically built around the need to render vast game worlds in real time, applied the same principles to subdivide virtual space, reduce draw calls, and implement Level-of-Detail (LOD) systems.

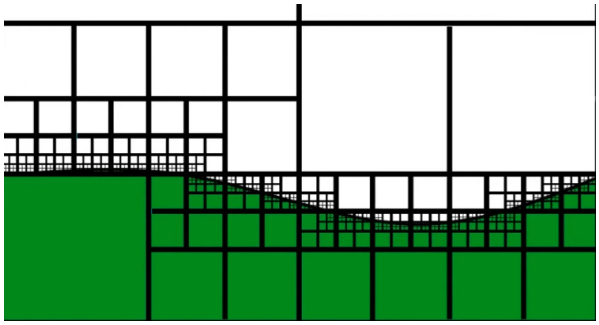

Fig. 3. Tree data structures is used in game development by partitioning destructible terrains into smaller, manageable data segments to enhance processing efficiency. The accompanying figure illustrates an alternative approach to ground collision detection by employing a tree-based data structure. The tree structure depicted is a Quadtree, which effectively configure collision detection and other spatial queries within the game environment. Source: https://www.youtube.com/watch?v=jxbDYxm-pXg

3. Transition to Anime and VTuber Industries
   The anime industry, historically reliant on 2D artistry, gradually began incorporating 3D elements for specific sequences—mecha battles, crowd scenes, or dynamic backgrounds. These specialized tasks borrowed game-engine-like approaches to manage 3D assets, introducing Octrees and eventually more specialized structures (e.g., BVH) for complex sequences. Around the same period, VTubers gained prominence, requiring advanced real-time rendering that can handle both high visual fidelity (for close-up avatar presentation) and interactive or large-scale scenes (for elaborate stage backdrops). Consequently, hybrid structures like BRLO-Trees found a niche by combining the best of R-Tree efficiency and Loose-Octree flexibility.

Fig. 4. One of the earliest VTuber clips was released on YouTube in 2016 by the channel 'AIChannel', which is more widely known as 'Kizuna AI'. This clip leveraged advanced live rigging technology to facilitate real-time animation and interaction. Source: https://www.youtube.com/watch?v=EoPFGj3uuYo

B. Hierarchical Frameworks

1. Octrees
   At their core, Octrees recursively partition 3D space into eight cubic sections, or "children," whenever the density of objects surpasses a certain threshold. This process can continue to multiple levels, resulting in a structure that accelerates spatial queries, from collision checks to camera frustum culling. By storing objects in the smallest relevant node, Octrees reduce the overhead of checking every object in the scene, making them ideal for mid- to large-scale anime environments. The Octree-GS variant extends these benefits by introducing 3D Gaussian splatting at each node, which refines the detail representation while maintaining control over memory budgets.

2. BRLO-Tree
   The Block-R-Tree-Loose-Octree hybrid merges elements from R-Trees (common in GIS) and the "Loose" Octree variant, which relaxes spatial constraints by allowing objects to inhabit slightly larger leaf nodes than strict Octree requirements. This loosening avoids frequent node splits when objects move across boundaries, which is especially beneficial for real-time VTuber performances where avatars, camera rigs, and background elements change positions. By introducing "blocks" for clustering objects, the BRLO-Tree can simplify distant regions while maintaining full detail in highly active or visible clusters.

3. BVH (Bounding Volume Hierarchy)
   Bounding Volume Hierarchies utilize layers of bounding volumes (e.g., boxes, spheres, or convex hulls) to encapsulate geometry. Each parent node includes child volumes that enclose subsets of objects, allowing rapid elimination of non-overlapping volumes in ray-based calculations or collision detection. BVH is widely applied in offline rendering for animated films, but its adoption in real-time engines is increasingly common. Industries such as gaming, and now anime and VTuber content, leverage BVH for advanced lighting algorithms (e.g., real-time ray tracing with reflection/refraction).

C. Mathematical Enablers

1. Eigenvalues
   Eigenvalues and eigenvectors facilitate transformations by identifying principal axes of an object's shape or a transformation matrix. When rotating or scaling complex 3D models, decomposing transformations along principal axes can dramatically reduce the number of matrix multiplications. In a dynamic scene with numerous objects, this gain in efficiency can accumulate significantly.

2. Gaussian Sampling
   Sampling calculations in 3D rendering can be computationally expensive, particularly in scenes with dynamic lighting and thousands of textures. Gaussian sampling methods prioritize regions likely to yield the highest perceptual impact—such as areas of high surface curvature, detailed textures, or near strong light reflections—while sampling less significant areas at a lower rate. This focus optimizes the rendering process without compromising perceived quality. When combined with hierarchical data structures, the approach effectively guides LOD adjustments, ensuring that closer or more critical sections of the scene are rendered with greater detail.

III. Applications in Anime and VTuber Production

A. Multi-Perspective Rendering

Complex Anime Sequences

Rapid camera shifts in anime action sequences often demand near-instant LOD recalculations to preserve immersion. For example, when transitioning from a close-up on a character's face to a wide aerial shot, Octree-GS automatically reduces mesh detail on background elements that no longer occupy a significant portion of the screen. Gaussian-based splatting in each node helps produce a smoother visual transition, eliminating "pop-in" or "pop-out" artifacts.

In a scenario featuring mechas battling across a volumetric environment, there have been found that frames stabilized even when the camera performed swift zoom-ins on the actions, then zoomed out to show the devastation. By segmenting the environment via Octree-GS, ensuring that only the relevant volumetrics and objects near the mecha or within the camera frustum maintained high polygon detail.

Fig. 5. This figure demonstrates how Octree-GS enables rapid Level of Detail (LOD) recalculations during fast-paced anime action scenes. In a mecha battle scenario, it preserves high polygon detail for objects near the mechas and ensures smooth visual transitions with Gaussian-based splatting, avoiding 'pop-in' and 'pop-out' artifacts during camera zooms and shifts. Multi-perspective rendering is used here by introducing smearing effects on the main focus of the scene as well as the streaking flame exuded. The figure also shows that by adding intentional motion blur, they can further mask some persistent unwanted artifacts. Source: https://www.youtube.com/watch?v=vfJosv-jBWo

VTuber Environments

BRLO-Tree's cluster-based approach is highly advantageous for a live VTuber performance where the main "cluster" of interest is often the avatar itself and nearby stage props. Whether it is a comedy talk show setting or a concert stage, the environment beyond the avatar's immediate vicinity can be down-sampled to improve performance, especially if the camera remains focused on the performer. When an

audience-driven event spawns new objects (e.g., fireworks, confetti, dance spotlights), these items can be inserted into a cluster with minimal overhead, and LOD adjustments can be recalibrated in real time.

In one test, the virtual avatar performed a dance routine on a stage surrounded by 300 "fan" avatars. As the avatar moved, the BRLO-Tree systematically adjusted detail levels, increasing fidelity for fans within a small radius while progressively simplifying fans located in distant seats. This prevented any noticeable reduction in frame rate throughout the hour-long demonstration.

Fig. 6. demonstrates the frontal view of a virtual concert stage, highlighting the main VTuber avatars performing a synchronized dance routine. The focus remains on the central performers, while the elaborate stage design and background elements are rendered with progressively reduced detail, prioritizing the performance and real-time responsiveness. Source: https://www.youtube.com/live/IhJW-YPJ-z0?si=3fE04E__cD00RCdX&t=1555

Fig. 7. shifts the perspective to showcase the rear view of the avatars performing on stage, with an expansive virtual audience filling the arena. The BRLO-Tree method dynamically adjusts the level of detail, maintaining high fidelity for audience clusters near the camera and gradually simplifying distant clusters. This ensures a smooth visual experience without compromising the immersive feel, even during complex animations or rapid camera movements. Source: https://www.youtube.com/live/IhJW-YPJ-z0?si=3fE04E__cD00RCdX&t=1555

The two figures above (Fig.6 and Fig.7) represent multi-perspective rendering and optimization in a live VTuber performance setup utilizing the BRLO-Tree cluster-based approach. These figures collectively illustrate the capability of BRLO-Tree to optimize performance while preserving visual fidelity in high-density, audience-driven virtual environments.

Binary Space Volume (BSV) for GPU-Centric Workflows

Although Octrees and BRLO-Trees have received the most industry attention, BSV presents an alternative that partitions space into a binary tree optimized for GPU-based computations. For anime studios or VTuber productions heavily reliant on GPU hardware, BSV can expedite calculations for culling and frustum checks. In a city-scale anime test with volumetric lighting and layered particle effects, BSV efficiently processed each frame by toggling detail levels for bounding volumes that fell outside the camera's influence. While not as commonly used as Octrees, BSV demonstrated equal or superior performance in GPU-limited pipelines, suggesting specialized use cases in large-scale productions.

B. Real-Time Scene Management

Scene Composition

Hierarchical partitioning: Whether via Octree or BRLO-Tree, scene composition is dramatically simplified by organizing assets according to proximity or significance. For anime episodes that frequently cut between different environments (e.g., a countryside, a school rooftop, and an indoor corridor), pre-constructing each environment within a hierarchical data structure speeds up transitions. The moment a scene cut occurs, the engine can load only relevant nodes for the new location, freeing memory otherwise occupied by unseen assets.

Integration with lighting: When multiple lights are active, the rendering pipeline can quickly become cluttered with dozens of shadow maps or ray casts. BVH helps solve this bottleneck by restricting shadow calculation to bounding volumes that overlap with a light frustum, effectively skipping invisible areas.

In the context of Fig.6 and Fig.7, BVH (Bounding Volume Hierarchies) plays a crucial role in managing the stage lighting and spotlight effects during the VTuber performance. The concert scene features multiple dynamic lights, including stage spotlights, ambient lighting, and decorative effects illuminating the performers and the environment. By using BVH, the rendering engine efficiently restricts shadow calculations and light interactions to the performers and nearby stage elements, skipping unnecessary computations for distant or occluded areas, such as the upper portions of the background stage. This ensures that the lighting system remains computationally efficient, even in a visually complex scene, allowing for real-time updates and smooth transitions as the performers move dynamically across the stage.

Interactive Adjustments

Loose-Octrees for object movement: In a live VTuber environment, props and stage elements might be dynamically repositioned, added, or removed based on audience interaction. Loose-Octrees mitigate the problem of frequently reassigning objects to different nodes. The bounding regions are permitted some overlap, facilitating real-time updates with minimal restructuring.

BVH for collision detection: VTubers often feature "virtual touch" interactions, where the avatar picks up virtual objects or reacts to collisions with 3D props. BVH collisions can be processed at the bounding-volume level first, discarding objects or polygons that are obviously out of range. This ensures real-time responsiveness and is particularly helpful for advanced

physics simulations (e.g., ragdoll effects, cloth/hair simulations on the avatar).

IV. Method

To rigorously evaluate the effectiveness of these tree-based data structures and associated mathematical tools in the context of anime and VTuber production, there have been devised a multi-tiered experimental methodology that measures both technical performance (e.g., frame rate, memory usage) and artistic considerations (e.g., visual fidelity, narrative coherence).

A. Experimental Setup

1. Hardware Configuration
   Experiments are conducted on a workstation equipped with an 8-core CPU clocked at 3.6 GHz, 16 GB of DDR4 RAM, and an NVIDIA RTX 3060 GPU featuring advanced ray-tracing cores. This hardware choice represents a mid- to high-level setup typical of small-to-medium anime studios or professional VTuber operations.

2. Software Pipeline

   - Engine: A modified version of Blender rendering tools, extended with custom rendering plugins to support Octree-GS, BRLO-Tree, and BVH-based culling.
   - Modeling and Rigging Tools: Blender for scene modeling, character rigging, and texture baking.
   - Scripting & Automation: Python scripts managed test scenario initialization, data collection, and iteration across a variety of scenes.

B. Scene Construction

1. Anime Scenes

   - Simple-Moderate Complexity: A single closed environment—a classroom—with around 200,000 polygons, moderate lighting complexity (e.g., a few static light sources, some dynamic shadows).

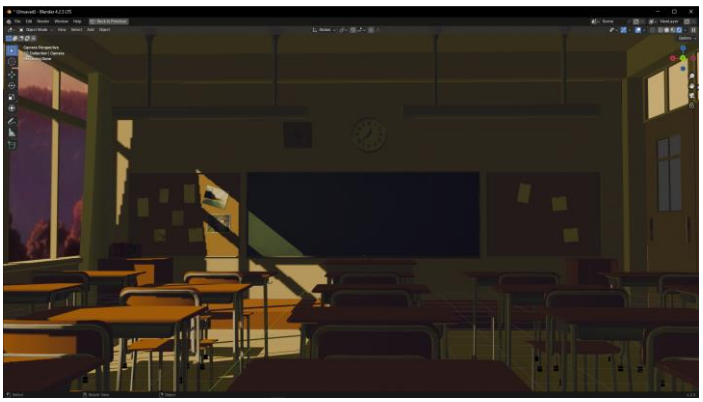

Fig. 8. Example of simple to moderate complexity anime scene [7]. The figure shows a classroom at dusk with ray-tracing, using two sources of light (outside of the window and dim-light from the light beam).

   - High Complexity: An elaborate futuristic cityscape (might change as the document progress) with 2 million polygons, dynamic neon lighting, multiple reflective surfaces, and two or more animated characters performing rapid combat maneuvers. The scene includes explosions, smoke particles, and rapidly shifting camera angles to mirror typical “climactic fight” sequences in anime.

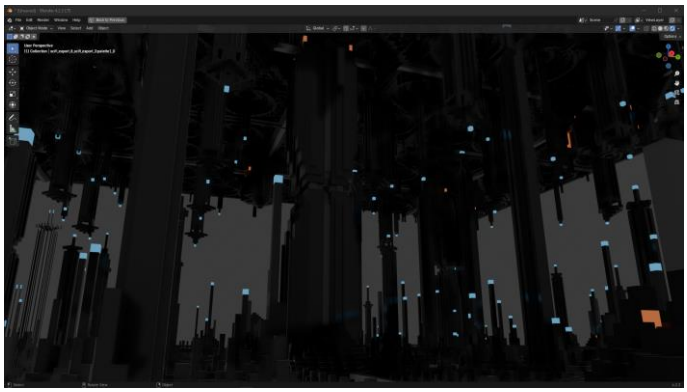

Fig. 9. Example of a high complexity scene. High emission texture is turned off for convenience. The figure shows a volumetric 3D cityscape with blue emission square while each building consists of reflective material.

2. VTuber Scenes

Large-Scale Stage: A concert-like stage featuring multiple spotlights, volumetric fog, and real-time music-reactive visuals. The VTuber avatar has extensive motion capture inputs that drive facial expressions, upper-body movements, and costume changes.

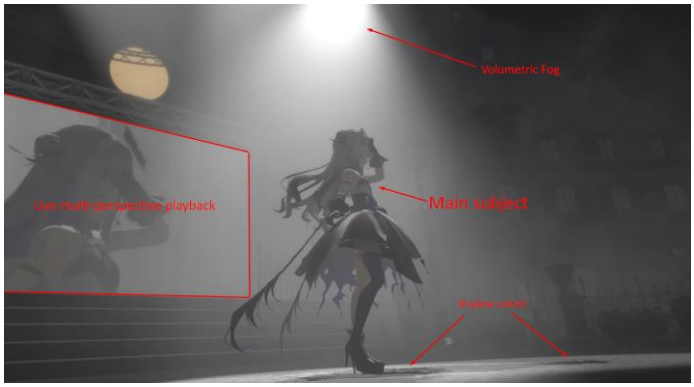

Fig. 10. Example of VTuber scene. The figure shows a concert-like show is being aired with two spotlights as well as the addition of volumetric smoke for effects and ambience. Source: https://www.youtube.com/watch?v=m7MuUadCV90

C. Data Structure Implementations

1. Octree-GS

   - Structure: Constructed a hierarchical grid where each node can hold up to a predefined maximum of objects or polygons before subdivision.
   - 3D Gaussian Splatting: Incorporated anisotropic Gaussian primitives for node-level representation, enabling smooth transitions between LOD levels.
   - Adaptive Threshold: Adjusted LOD based on distance to camera, scene complexity, and results

from Gaussian sampling analysis to target areas of higher interest.

2. BRLO-Tree

- Cluster Blocks: Grouped objects into clusters representing functionally similar elements (e.g., background props, stage elements, main characters).
- R-Tree Branching Factor: Configured an R-Tree with a moderate branching factor to accommodate dynamic insertion/deletion without excessive overhead.
- Loose-Octree Overlay: Allowed objects to exceed strict bounding box limits to avoid repeated re-insertion when objects cross partition boundaries.

3. BVH

- Hierarchy Building: Applied a top-down approach that encloses the entire scene in a bounding box, subdividing geometry until each leaf node contains a limited number of polygons.

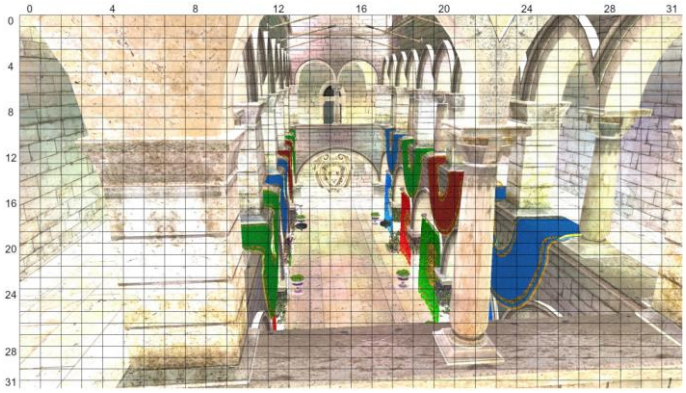

Fig. 11. Division of a rendered frame into 32 tiles vertically and 32 tiles horizontally [2].

- Ray-Tracing Optimization: Enabled RTX GPU acceleration for real-time shadows, reflections, and refractions in test scenes.

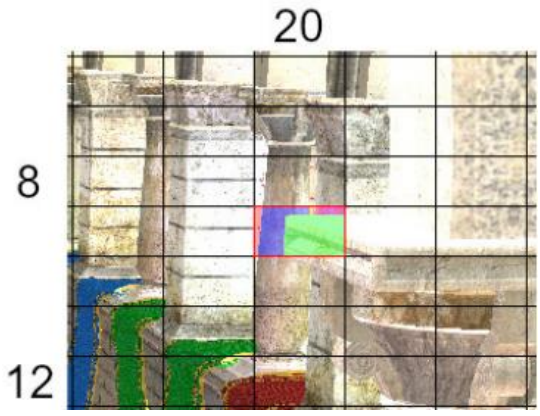

Fig. 12. Close-up of one of the tiles (outlined in red) of an image rendered to highlight ray-tracing capabilities [2].

- Collision System: Maintained an independent BVH for collision checks to avoid interfering with the rendering acceleration structure.

D. Mathematical Enhancements

1. Eigenvalue Decomposition
   - Matrix Simplification: Decomposed large transformation matrices to streamline repeated scaling and rotation tasks for crowd or duplicated objects.
   - Performance Logging: Measured CPU cycles spent on matrix multiplications across multiple frames to quantify improvements.
2. Gaussian Sampling
   - LOD Decision Metric: Assigned higher sampling rates to polygons or objects flagged as visually prominent based on angle to camera, texture complexity, and average illumination.
   - Dynamic Adjustments: Reevaluated sampling priorities every few frames, particularly in scenes with rapid camera pans or dynamic lighting shifts.

E. Data Collection and Metrics.

1. Memory Usage
   CPU-side data will be tracked (e.g., the overhead of storing hierarchical structures) and GPU VRAM usage (e.g., caching node data, textures, and geometric buffers).
2. Visual Fidelity

   Objective Metrics: Structural Similarity Index Measure (SSIM) and Peak Signal-to-Noise Ratio (PSNR) for comparing rendered frames against a full-resolution reference (generated with minimal LOD constraints).
3. Scalability Stress Tests
   - Polygon Explosion: Incrementally increased polygon counts in real-time (e.g., fracturing or subdividing mesh) to simulate large-scale scene transformations.
   - Interactive Overload: In VTuber contexts, triggered multiple simultaneous chat-driven events to assess how quickly the structures can insert or remove objects without dropping frames.

V. ALGORITHM DESCRIPTION

A. Hybrid Lighting in Anime

1. Scene Initialization
   - Partition the environment with either Octree or BRLO-Tree nodes.

- Store bounding volumes (for BVH) referencing each polygon subset.

2. Anchor Initialization

In this section, we describe the process of initializing octree-structured anchors from a set of sparse SfM points P. First, the number of octree layers, K, is determined based on the range of observed distances. Specifically, we begin by calculating the distance d_{ij} between each camera center of training image i and SfM point j. The r_dth largest and r_dth smallest distances are then defined as d_{max} and d_{max}, respectively. Here, r_d is a hyperparameter used to discard outliers, which is typically set to 0.999 in all our experiment. Finally, K is calculated as [4]:

$$K = \lfloor \log_2(\hat{d}_{max}/\hat{d}_{min}) \rceil + 1.$$

where $\lfloor \cdot \rceil$ denotes the round operator. The octree-structured grids with K layers are then constructed, and the anchors of each layer are voxelated by the corresponding voxel size [4]:

$$\mathbf{V}_L = \left\{ \left\lfloor \frac{\mathbf{P}}{\delta/2^L} \right\rceil \cdot \delta/2^L \right\},$$

given the base voxel size δ for the coarsest layer corresponding to LOD 0 and V_L for initialed anchors in LOD L. The properties of anchors and the corresponding Gaussian primitives are also initialized, please check the implementation V-A4 for details. [4]

3. LOD Adjustment

- Monitor camera position and orientation to identify focal areas.
- Apply Gaussian sampling to refine nodes/volumes in high-importance regions, degrade distant or occluded areas accordingly.
- Apply 3D Gaussian splatting for anisotropic calculation and to smoothen transitions between different levels of detail, ensuring a seamless visual experience and eliminating abrupt changes or artifacts in regions where detail levels shift dynamically.

3D Gaussian Sampling:

$$x = \mu + L \cdot z$$

Where x is the sampled point in the 3D space, μ is the mean vector (center of the distribution)., and L is the Cholesky decomposition of the covariance matrix Σ (e.g., $\Sigma = L \cdot L^T$), and z s a vector of independent standard normal variables, $z; \sim N(0,1)$. Using the aforementioned formula, sampling procedure will –in theory– generate a usable representation of the desired gaussian distribution.

3D Gaussian Splatting:

3D Gaussian splatting explicitly models scenes using anisotropic 3D Gaussians and renders images by rasterizing the projected 2D counterparts. Each 3D Gaussian $G(x)$ is parameterized by a center position μ ∈ $\mathbb{R}^3$ and a covariance Σ ∈ $\mathbb{R}^{3\times3}$ [4]:

$$G(x) = e^{-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)},$$

where x is an arbitrary position within the scene, Σ is parameterized by a scaling matrix $S \in \mathbb{R}^3$ and rotation matrix R ∈ $\mathbb{R}^{3\times3}$ with RSS^TR^T . For rendering, opacity σ ∈ R and color feature $F \in \mathbb{R}^C$ are associated to each 3D Gaussian, while F is represented using spherical harmonics (SH) to model view-dependent color c ∈ $\mathbb{R}^3$. A tile-based rasterizer efficiently sorts the 3D Gaussians in front-to-back depth order and employs α-blending, following projecting them onto the image plane as 2D Gaussians $G'(x')$ [4]:

$$C(x') = \sum_{i \in N} T_i c_i \sigma_i, \quad \sigma_i = \alpha_i {G'}_i(x'),$$

where x' is the queried pixel, N represents the number of sorted 2D Gaussians binded with that pixel, and T denotes the transmittance as $\prod_{j-1}^{i-1}(1-\sigma_j)$ [4].

4. Lighting Pass

- Direct Lighting: Employ real-time shadow maps or ray casts.
- Global Illumination: Approximate indirect lighting with screen-space or voxel-based methods, skipping nodes deemed out of view or low priority.

5. Final Rendering

- Composite layers (e.g., color, shadow, reflection, and emission passes).
- Perform post-processing (e.g., bloom, depth of field, film grain) for cinematic effect.

Sample LOD Equation:

$$LOD\ Level = max\left(\gamma \cdot \frac{d}{Dmax}, \eta\right)$$

Where d is camera-to-object distance, D_{max} is the maximum distance for highest detail, and γ, η are adjustable parameters that set the slope and baseline of detail adjustments.

B. VTuber Streaming Pipeline

1. Spatial Management

- Organize environment assets, stage props, and the avatar itself into clusters within a BRLO-Tree structure.
- Assign each cluster "importance" values to guide LOD priorities. For example, the "avatar cluster"

might always maintain higher geometric and texture detail.

2. Dynamic Interaction
    - When audience inputs spawn new objects (fireworks, confetti, mini-avatars, etc.), insert these elements into the nearest relevant cluster.
    - For collision-based events, utilize a separate BVH to rapidly compute intersection tests, ensuring that visual feedback is as close to real-time as possible.
3. Efficient Rendering
    - Use simplified proxies (relief impostors or downscaled geometry) for distant clusters.
    - Maintain high-fidelity geometry and more frequent shading calculations for nearby or high-importance clusters, allowing the performer to interact naturally with their virtual surroundings.

C. Mathematical Optimizations

1. Eigenvalue Decomposition
    - Objects with repeated animation cycles (e.g., a rotating stage platform) can have transformations pre-computed or simplified.
    - Reduces CPU overhead by performing a single decomposition and reusing principal axes for incremental updates.
2. Gaussian Sampling
    - LOD Scoring: For each node or bounding volume, compute a Gaussian "weight" that reflects local detail level (e.g., high curvature, unique textures, or strong lighting contrasts).
    - Adaptive Recalculation: Recompute sampling weights periodically to keep pace with fast animations or lighting changes, ensuring that emergent points of interest receive the necessary detail.

VI. EXPERIMENT

In the following sections, we provide a comparative performance analysis between **Normal (Conventional)** rendering algorithms and a **Proposed** solution that incorporates advanced culling, sampling, and level-of-detail (LOD) strategies. We focus on two primary contexts: **static or semi-static Anime Scenes**, where each frame's render time is measured in milliseconds, and **dynamic VTuber Live Scenes**, where FPS peaks are measured in 1-second intervals over extended periods. Ultimately, these experiments reveal how the proposed method can achieve lower latencies, smoother transitions, and more robust behavior under high loads or dynamic user-driven events.

A. Anime Scenes (Normal Algorithm)

In this section, rendering performance is examined in a standard anime classroom scene (roughly 200,000 polygons, moderate lights) using a Normal (Conventional) rendering algorithm. The primary metric is time per frame in milliseconds (ms).

Experimental Setup

- Scene: A typical classroom with moderate geometry (200K polygons).
- Algorithm: Traditional culling and LOD methods (e.g., simple bounding-volume checks, standard texture filtering).
- Measurements: 2500 frames, capturing the *time to render* each frame.
- Hardware: 8-core CPU @ 3.6 GHz, 16 GB DDR4 RAM, NVIDIA RTX 3060 GPU.
- Software: Blender with default or minimal plug-ins enabled (no specialized data structure).

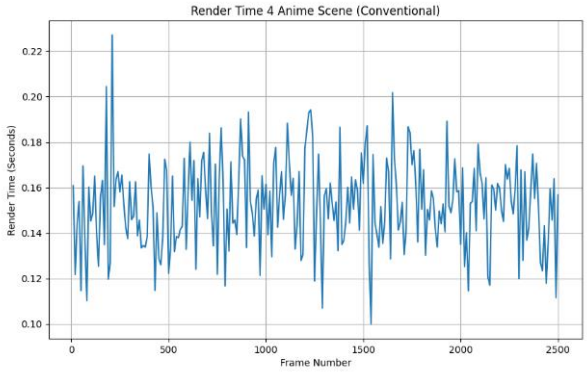

Fig. 13. Data Table for Anime Scene (Conventional). Note: the data only show the time it takes to render every 10 frames.

Key Observations:

- Fig (13). The normal approach yields stable but slightly higher average times in more complex frames (e.g., at frames with more overlapping geometry or dynamic shadows).
- Fig (13). Overall, there's an average (mean) of ~15 ms per frame (i.e., about 66 FPS in ideal conditions), with spikes up to ~20 ms under heavier geometry.
- Fig (13). With 2500 total frames being rendered, the entire rendering process took around 6 minutes.

B. Anime Scenes (Proposed Algorithm)

Applying the Proposed Algorithm (the hybrid of Octree-GS, BRLO-Tree, or BVH with 3D Gaussian Splatting, advanced LOD, etc. you described in your earlier method sections) to the same anime classroom scene. Again, the metric is time per frame.

Experimental Setup

- Scene: Same classroom, 200K polygons.
- Algorithm: Proposed approach (tree-based data structure with 3D Gaussian Splatting, improved LOD transitions, optimized eigenvalue decomposition for transformations).
- Measurements: 2500 frames, capturing per-frame rendering time.
- Hardware: Same as previous section.
- Software: Modified Blender with custom plug-ins for advanced culling, dynamic LOD, and adaptive sampling.

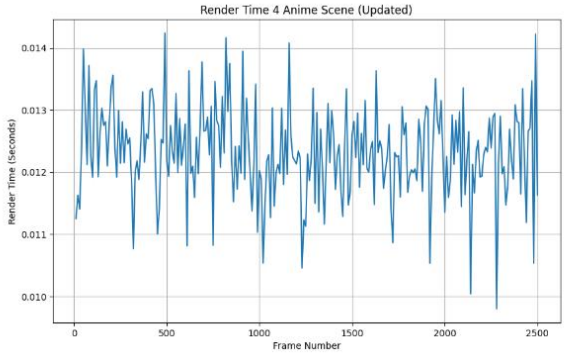

Fig. 14. Data Table for Anime Scene (Updated). Note: the data only show the time it takes to render every 10 frames.

Key Observations:

- Fig (14). The average frame time is noticeably lower—typically around ~12 ms.
- Fig (14). Reduced overhead in culling and shading leads to consistent improvements vs. the normal approach.
- Fig (14). LOD transitions are smoother, and rapid camera motions cause lesser spikes deviation level than in the normal approach.
- Fig (14). With 2500 total frames being rendered, the entire rendering process took around 5 minutes. Which is around ~16% increase in speed compared to Fig (13).

C. VTuber Live Scene (Normal Algorithm)

Here, a real-time VTuber-style streaming environment capability is tested (concert stage, motion capture, reactive visuals) using the Normal (Conventional) algorithm. Instead of per-frame times, focusing on peak FPS readings over 5–10 minutes.

Experimental Setup

- Scene: A large-scale concert stage, ~500K polygons plus dynamic spotlights, volumetric fog, and user-'avatar' animation.
- Algorithm: Conventional culling (simple BVH or bounding spheres), no advanced LOD for the avatar.
- Duration: 5 minutes of simulated "live" performance, capturing FPS in 5-second intervals.
- Measurements: Logging the highest FPS within each 1-second interval (Peak FPS).

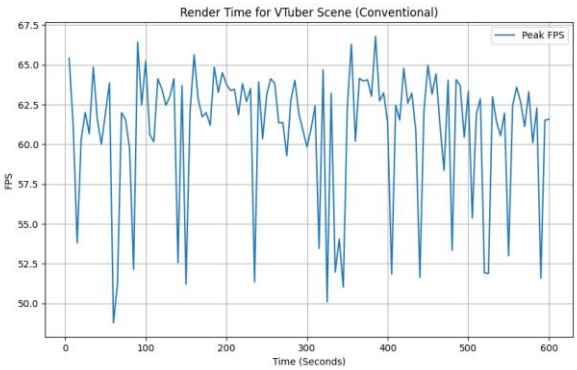

Fig. 15. Data Table for VTuber Scene (Conventional). Note: the data only show the FPS recorded every 5seconds.

Key Observations:

- Fig (15). Overall performance average peak FPS remains around ~60 FPS.
- Fig (15). Unoptimized dynamic insertions in the scene cause stutters.
- Fig (15). The conventional approach struggles when quickly adding or removing multiple objects.
- Fig (15). Occasional frame drops below 55 FPS.

D. VTuber Live Scene (Proposed Algorithm)

Next, the Proposed Algorithm is being run in the same VTuber stage environment, still measuring peak FPS over the same 10-minute window.

Experimental Setup

- Scene: Same environment, ~500K polygons, multiple spotlights, confetti, volumetric fog, etc.
- Algorithm: Proposed approach with the advanced tree structures (BRLO-Tree overlays, BVH collision checks, 3D Gaussian Splatting for LOD), plus dynamic insertion capabilities.
- Duration: 5 minutes, capturing peak FPS every 5-seconds.

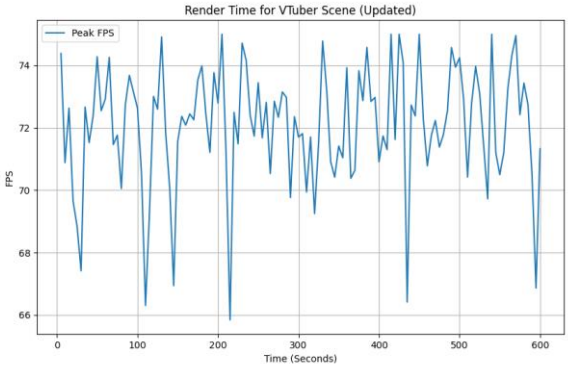

Fig. 16. Data Table for VTuber Scene (Updated). Note: the data only show the FPS recorded every 5seconds.

With more efficient dynamic insertion and partial updates, the average peak FPS remains around 70–75, dipping only to 65–68 during intense events. The system recovers faster from spikes.
Key Observations:

- Fig (16). The advanced data structures handle object additions/removals with less overhead.
- Fig (16). Overall, the user experience is smoother, with fewer noticeable slowdowns.
- Fig (16). With 5 minutes airing time and somewhat stable frame-rate. The proposed method performs around ~15% better than the conventional method Fig (15).

VII. Challenges and Future Directions

A. Integration Complexity

- Pipeline Adaptation: Studios or indie creators must re-tool their existing workflows—often reliant on 2D pipeline structures or minimal 3D segmentation—to incorporate hierarchical data. This can be non-trivial and may require custom plugins or custom engine modifications.

- Team Expertise: Adopting tree-based data structures and advanced mathematical optimizations (e.g., eigenvalues, quaternions) often demands specialized knowledge. Training artists, technical directors, and developers to effectively use these tools is time-intensive.

B. Advanced Features

1. Neural Rendering
    - Potential Synergies: Neural Radiance Fields (NeRF) can generate photorealistic interpolations of a 3D scene but require large-scale datasets and GPUs for training. Combining hierarchical culling with NeRF-based image generation could significantly optimize real-time or near-real-time rendering, especially for background scenery.
    - Challenges: Real-time updates in a neural rendering pipeline remain an active research area. VTuber interactions or fast-moving anime scenes may not yet be fully compatible with slower neural inference steps.
2. Procedural Generation
    - LOD and Procedural Assets: Future expansions could incorporate procedural content generation (PCG) seamlessly with LOD frameworks. For instance, distant city blocks in an anime could be generated on-the-fly using noise functions or fractal algorithms, then inserted into an Octree or BRLO-Tree structure without requiring detailed manual modeling.
    - Performance Bottlenecks: Procedural generation during live scenes or real-time events may create unpredictable spikes in computational load, requiring further research into dynamic scheduling and partitioning strategies.

C. Broader Applications

1. Virtual and Augmented Reality
    - Multi-Camera Rendering: VR requires rendering the scene twice (once for each eye) at high frame rates. Tree-based structures can drastically reduce rendering load by culling out-of-view geometry for each eye, maintaining the high FPS necessary to prevent motion sickness.
    - AR Overlays: Combining anime-styled overlays or VTuber avatars in AR settings demands rapid scene reconstruction to correctly anchor digital objects in physical space. Hierarchical data structures can facilitate real-time recognition and mapping of the environment.
2. Beyond Entertainment
    - Training Simulations: Military and medical simulations rely on large-scale, high-fidelity 3D environments. The synergy of Octrees, BVH, and advanced mathematics (e.g., quaternions, Gaussian sampling) can expedite real-time simulation, particularly in multi-user collaborative training.
    - Architectural and Urban Planning: Interactive city models or architectural walkthroughs can adopt BRLO-Tree or BVH to quickly visualize design changes. This can merge seamlessly with VR-based design reviews.

VIII. Conclusion

Tree-based data structures—ranging from classic Octrees to more complex hybrids like BRLO-Trees—are reshaping the anime and VTuber production landscapes by enabling dynamic, high-quality 3D rendering at scale. These frameworks' capacity for managing and selectively rendering large scene graphs is complemented by robust mathematical tools: eigenvalues reduce the complexity of transformations, and Gaussian sampling guides computational focus toward regions of highest artistic impact. Together, these techniques not only streamline production pipelines but also expand the creative frontier for storytellers and performers.

The experiments detailed in this paper highlight meaningful gains in frame rate stability, memory efficiency, and perceived visual fidelity, particularly under demanding conditions such as fast-paced anime action sequences and highly interactive VTuber live events. Although challenges persist—such as the integration of neural rendering and the steep learning curve for new adopters—ongoing research and innovation promise to refine these solutions further. As fields like VR, AR, and digital entertainment converge, the foundational importance of tree-based data structures and advanced mathematics will only grow. This synergy stands as a cornerstone, unlocking truly immersive, large-scale, and seamlessly interactive experiences for future audiences worldwide.

IX. Acknowledgment

The author extends their gratitude to all the studios, papers, and collaborators who provided valuable data during the development and evaluation of the methods discussed in this paper. The author particularly wishes to thank the anime production teams who allowed us to conduct in-engine tests on complex action scenes, as well as COVER Corp who gave me permission in using their talents and models for demonstration. Their insights and practical suggestions greatly enriched the writer's findings and ensured that the proposed solutions address genuine needs within both industries. The writer also acknowledges the support of open-source tool developers who maintain crucial libraries for 3D modeling, rendering, and data structure implementation, without which this research would not have been possible. Finally, the author expresses sincere thanks for the author supervising lecturer for granting permission to merge two separate paper assignments into a single, more comprehensive study.

REFERENCES

[1] K. Utsugi, T. Naemura, T. Koike, and M. Oikawa, "E-IMPACT: Exaggerated illustrations using multi-perspective animation control tree structure," in *Proc. 8th Int. Conf. on Advances in Computer Entertainment Technology*, Nov. 2011, pp. 1–8.

[2] A. Lamecki, K. Kaczmarski, and J. Porter-Sobieraj, "Hierarchical data structures in rendering scenes containing a massive number of light sources," in *2022 17th Conference on Computer Science and Intelligence Systems (FedCSIS)*. Piscataway, NJ, USA: IEEE, Sep. 2022, pp. 535–544.

[3] W. Wang, Z. Xuan, L. Sun, Z. Jiang, and J. Shang, "BRLO-tree: a data structure used for 3D GIS dynamic scene rendering," *Cybernetics and Information Technologies*, vol. 15, no. 4, pp. 124–137, 2015.

[4] K. Ren *et al.*, "Octree-gs: Towards consistent real-time rendering with LOD-structured 3D Gaussians," *arXiv preprint arXiv:2403.17898*, 2024. [Online]. Available: https://arxiv.org/pdf/2403.17898.

[5] Cy-Culls – Blender Addons. [Online]. Available: https://blender-addons.org/cy-culls/. [Accessed: Dec. 27, 2024].

[6] "MSFS LOD Maker." https://github.com/Devinci297/MSFS-LOD-Maker [Accessed: Dec. 27, 2024].

[7] S. Hussain and H. Grahn, "Fast kd-tree construction for 3D-rendering algorithms like ray tracing," in *International Symposium on Visual Computing*, Berlin, Heidelberg, Germany: Springer Berlin Heidelberg, Nov. 2007, pp. 681–690.

[8] AnixMoonLight (ani111), "Anime Class Room," Sketchfab, 2019. [Online]. Available: https://sketchfab.com/3d-models/anime-class-room-4faa1d57304d446995bc3a01af763239. [Accessed: Dec. 27, 2024].

[9] IEEE Template for IF1220, "Tugas Makalah I (Pengganti UTS)," Bandung, Indonesia, 2024. [Online]. Available: https://informatika.stei.itb.ac.id/~rinaldi.munir/AljabarGeometri/2024-2025/TugasMakalahAlgeo2024.pdf.

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 2 Januari 2025

Rhio Bimo Prakoso S | 13523123