

# Cluster-Base Level of Detail Technique Analysis

## Periodic Report

Yang Chen

Student ID: 20816397

User ID: y2588che

December 3, 2024

## Abstract

This report presents the development of a real-time ClusterLOD Renderer designed for efficient rendering of large-scale scenes with complex geometry. The project addresses challenges in maintaining high visual fidelity, optimizing performance, and reducing memory usage, leveraging modern GPU techniques. Key methods include mesh partitioning, Quadric Error Metrics (QEM) simplification.

The motivation for this work stems from significant advancements in game engine technologies in recent years, particularly the introduction of Nanite and Lumen in Unreal Engine 5 in 2020. Nanite revolutionized mesh handling by enabling the efficient loading and use of massive geometry data in real-time scenes, while Lumen focused on indirect global illumination, delivering realistic lighting and reflections. While Lumen improves visual quality by simulating light transport, this project emphasizes addressing the challenges of rendering massive geometry datasets, akin to those targeted by Nanite.

This project primarily focuses on managing large-scale meshes by utilizing clustering techniques to achieve two core objectives: first, to parallelize and accelerate the QEM simplification algorithm, enabling efficient precomputation of mesh levels of detail; second, to allow piecewise Level of Detail (LOD) selection instead of traditional whole-mesh LOD selection. These innovations make it possible to dynamically adjust detail levels across different parts of a scene, optimizing both performance and memory usage in real-time applications.

By combining clustering, hierarchical data structures, and a GPU-driven pipeline, this work advances the state-of-the-art in real-time rendering, making it well-suited for applications requiring high-quality visuals and efficient resource utilization.

# Contents

<b>1</b>	<b>Introduction</b>	<b>5</b>
1.1	Purpose . . . . .	5
1.2	Goal . . . . .	5
<b>2</b>	<b>Background and Related Work</b>	<b>6</b>
2.1	Existing Techniques . . . . .	6
2.2	Relevant Concepts . . . . .	7
2.2.1	Mesh Partitioning . . . . .	8
2.2.2	QEM Simplification . . . . .	8
2.2.3	Bounding Volume Hierarchy (BVH) . . . . .	10
2.2.4	Half Edge Data Structures . . . . .	10
<b>3</b>	<b>Methodology</b>	<b>10</b>
3.1	Precomputation Steps . . . . .	10
3.2	Runtime Pipeline . . . . .	10
3.3	Post-Processing . . . . .	10
<b>4</b>	<b>Implementation</b>	<b>11</b>
4.1	Tools and Libraries . . . . .	11
4.2	Algorithms and Data Structures . . . . .	11
4.3	Optimizations . . . . .	11
<b>5</b>	<b>Results and Analysis</b>	<b>11</b>
5.1	Visual Output . . . . .	11
5.2	Performance Metrics . . . . .	11
5.3	Quality Metrics . . . . .	11
<b>6</b>	<b>Discussion</b>	<b>11</b>
6.1	Achievements . . . . .	11
6.2	Limitations . . . . .	11
6.3	Future Work . . . . .	12
<b>7</b>	<b>Conclusion</b>	<b>12</b>
<b>8</b>	<b>Bibliography</b>	<b>12</b>

<b>A</b>	<b>Appendices</b>	<b>12</b>
A.1	Code Samples . . . . .	12
A.2	Additional Results . . . . .	12
A.3	Glossary . . . . .	12

# 1 Introduction

## 1.1 Purpose

The motivation for this project originates from the impressive advancements showcased by Unreal Engine 5 (UE5) in 2020, particularly the Tomb Raider Cave demo. This demo demonstrated the capability to render billions of triangles in a single frame, all in real-time. Inspired by this, I aim to develop a similar system capable of rendering large-scale scenes in real-time, enabling the creation of more complex and immersive environments in games, ultimately enhancing realism.

While UE5's Lumen focuses on real-time global illumination by combining multiple global illumination algorithms, it works by accumulating radiance over time. It employs a 'render-SurfaceCache' to convert to and from VoxelSH spheres, achieving real-time results through temporal radiance accumulation. This allows Lumen to deliver highly realistic lighting in dynamic scenes.

Our project, however, focuses on Nanite, another groundbreaking technology introduced alongside Lumen. Nanite addresses the challenge of efficiently rendering vast numbers of triangles in large-scale scenes. Its core innovation lies in the ability to quickly select visible triangles and stream them into the GPU. This is essential for scenarios where scenes contain a high density of triangles but are viewed from a distance. In such cases, the resolution of the view often cannot match the scene's geometric detail, leading to significant overdraw, as multiple triangles may be rendered for a single pixel.

To solve this, an efficient Level of Detail (LOD) system is required. The LOD system dynamically adjusts the level of detail based on the viewpoint, reducing overdraw and improving rendering performance. This project aims to study and replicate these ideas, exploring how to implement a robust and efficient ClusterLOD rendering system.

## 1.2 Goal

The goal of this project is to develop a real-time ClusterLOD renderer for efficiently rendering large-scale scenes with complex geometries.

This problem can be divided into two main parts: precomputation and runtime pipeline.

In the **precomputation** stage, the tasks include:

1. **Mesh Partitioning:** Divide the mesh into multiple clusters, where each cluster represents a group of spatially adjacent triangles.
2. **QEM Simplification:** Simplify the clusters using the Quadric Error Metrics (QEM) algorithm, reducing the number of triangles while maintaining visual fidelity.
3. **LOD Hierarchy Construction:** Repeat the above process iteratively to generate a Level of Detail (LOD) hierarchy. Finally, build a Bounding Volume Hierarchy (BVH) to organize these LODs, enabling fast, compute-shader-based selection and streaming.

In the **runtime pipeline**, the tasks include:

1. **LOD Selection and Streaming:** Select and stream visible clusters dynamically based on the view frustum and screen-space error metrics.
2. **Visibility Buffer Implementation:** Implement a visibility buffer to assist with cluster culling. Inspired by UE5, this involves using the visibility buffer from the previous frame to conservatively determine visible objects, followed by further culling calculations for remaining clusters.
3. **Deferred Shading Pipeline:** Integrate the visibility buffer into the deferred shading pipeline without disrupting its workflow.
4. **Real-time Effects:** Implement advanced real-time effects, including Screen Space Reflections (SSR), Screen Space Ambient Occlusion (SSAO), and Multi-Sample Anti-Aliasing (MSAA).
5. **Post-Processing Effects:** Apply effects such as bloom, tone mapping, and color grading to enhance the final rendered image.

By addressing both precomputation and runtime challenges, this project aims to create a high-performance ClusterLOD rendering system capable of handling the complexities of large-scale, detailed scenes in real-time.

## 2 Background and Related Work

### 2.1 Existing Techniques

Level of Detail (LOD) rendering is a critical technique for managing performance in real-time rendering. Various LOD methods have been developed to balance visual fidelity and

computational cost. This section provides an overview of key LOD techniques:

1. **Discrete Level of Detail (DLOD):** Objects are preprocessed to generate multiple resolutions, such as high, medium, and low LODs. The appropriate LOD is selected at runtime based on the viewer’s distance. While simple and efficient, this method may suffer from popping artifacts during transitions. This is generally generated by the artist and is not dynamic.
2. **Continuous Level of Detail (CLOD):** Instead of predefined levels, geometry is simplified dynamically in real-time, ensuring smooth transitions. This approach is commonly used in terrain rendering, such as the ROAM algorithm, but requires higher computational effort.
3. **Screen Space Error Metrics:** This method uses the projected size of objects on the screen to dynamically adjust LOD. By measuring pixel coverage, this technique ensures optimal resource allocation and minimizes overdraw.
4. **Cluster-Based LOD:** Introduced in Unreal Engine 5 as Nanite, this technique divides models into clusters and generates multi-level LODs. GPU-based parallel processing enables fast selection and streaming of visible clusters, making it suitable for large-scale, highly detailed scenes.
5. **Image-Based LOD (Billboarding):** Objects far from the camera are replaced with 2D textures mapped onto planes, significantly reducing geometry complexity. This technique is widely used for rendering vegetation and distant objects in large scenes.
6. **Geometry Shaders for LOD:** Geometry shaders dynamically adjust the level of detail during rendering. Although flexible, this approach may face performance bottlenecks for highly detailed models.

Each of these techniques offers unique advantages and trade-offs. In this project, we focus on Cluster-Based LOD to handle large-scale scenes with high geometric complexity, inspired by Nanite’s efficient cluster-based streaming and selection pipeline.

## 2.2 Relevant Concepts

This section provides an overview of the key concepts used in this project: mesh partitioning, QEM simplification, and Bounding Volume Hierarchy (BVH).

### 2.2.1 Mesh Partitioning

Mesh partitioning is the process of dividing a mesh into smaller, spatially coherent clusters. This is essential for efficient processing and management of complex geometries in real-time rendering. One common approach is to use graph partitioning algorithms, such as those implemented in the METIS library. METIS works by representing the mesh as a graph, where vertices correspond to mesh elements (e.g., triangles) and edges represent adjacency relationships. By minimizing the edge cut between partitions, METIS ensures that clusters are spatially cohesive, which is critical for Level of Detail (LOD) systems and BVH construction.

### 2.2.2 QEM Simplification

Quadric Error Metrics (QEM) simplification is a widely-used algorithm for reducing the complexity of a mesh while preserving its visual fidelity. The core idea is to iteratively collapse edges by combining two vertices into one, minimizing the geometric error introduced by the operation. This section describes the steps of the QEM simplification algorithm and formulates the edge collapse operation mathematically.

#### Steps of QEM Simplification:

1. **Initialize Error Matrices:** For each vertex  $v_i$  in the mesh, calculate its associated quadric error matrix  $Q_i$ . This matrix is defined based on the sum of the squared distances to the planes of all triangles adjacent to  $v_i$ :

$$Q_i = \sum_{p \in \text{adjacent planes}} \mathbf{k}_p \mathbf{k}_p^T$$

where  $\mathbf{k}_p = [a, b, c, d]^T$  is the plane equation  $ax + by + cz + d = 0$  normalized for each triangle plane adjacent to  $v_i$ . This 4 by 4 matrix can be used to calculate the distance from a point to a plane derived from [1].

2. **Select Candidate Edges:** Identify all edges in the mesh that are candidates for collapsing. An edge is defined by two vertices  $v_i$  and  $v_j$ .
3. **Compute Collapse Cost:** For each edge  $(v_i, v_j)$ , calculate the combined quadric error matrix:

$$Q_{ij} = Q_i + Q_j$$



The optimal new position  $v'$  is computed by minimizing the quadric error:

$$\text{Error}(v') = v'^T Q_{ij} v'$$

where  $v' = [x, y, z, 1]^T$ . To find the position  $v'$  that minimizes this error, the following steps are taken:

(a) Represent  $Q_{ij}$  as:

$$Q_{ij} = \begin{bmatrix} A & b \\ b^T & c \end{bmatrix}$$

where  $A$  is a  $3 \times 3$  matrix,  $b$  is a  $3 \times 1$  vector, and  $c$  is a scalar.

(b) Solve for  $v'$  if  $A$  is invertible:

$$v' = -A^{-1}b$$

(c) If  $A$  is singular (not invertible), choose  $v'$  as one of the following:

- Vertex  $v_i$ ,
- Vertex  $v_j$ ,
- The midpoint of  $(v_i, v_j)$ :  $\frac{v_i + v_j}{2}$ .

Compare the quadric error values at these candidate positions and select the one with the smallest error.

4. **Edge Collapse Error:** The quadric error associated with collapsing an edge  $(v_i, v_j)$  is defined as:

$$\text{Error}(v') = v'^T Q_{ij} v'$$

where  $Q_{ij}$  is the combined quadric matrix of  $v_i$  and  $v_j$ . This error represents the deviation of the simplified surface from the original.

5. **Collapse Edge:** Collapse the edge  $(v_i, v_j)$  into the new vertex  $v'$ . Update the mesh connectivity and recompute affected triangles to reflect the collapse.
6. **Repeat Until Target Simplification:** Continue collapsing edges with the lowest cost until the target number of triangles or the desired error threshold is reached.

By iteratively applying the steps described above, QEM simplifies the mesh while maintaining its essential geometric features, making it an ideal choice for real-time LOD rendering.

### 2.2.3 Bounding Volume Hierarchy (BVH)

A Bounding Volume Hierarchy (BVH) is a tree-like data structure used to organize objects in a 3D space. Each node in the BVH contains a bounding volume (e.g., an axis-aligned bounding box) that encapsulates a subset of the scene’s geometry. BVHs are widely used in real-time rendering for tasks like frustum culling, ray tracing, and LOD selection. By hierarchically grouping objects, BVHs enable efficient spatial queries and reduce the computational cost of rendering. In the context of ClusterLOD, BVHs are constructed to organize LOD levels, allowing for fast selection and streaming of visible clusters based on the view frustum and screen-space error metrics.

These concepts—mesh partitioning, QEM simplification, and BVH—form the foundation of the ClusterLOD system implemented in this project, enabling efficient processing and rendering of large-scale scenes.

### 2.2.4 Half Edge Data Structures

## 3 Methodology

### 3.1 Precomputation Steps

1. Mesh Partitioning with METIS.
2. Quadric Error Metrics (QEM) Simplification.
3. Construction of the LOD Hierarchy.
4. BVH Construction for LOD levels.

### 3.2 Runtime Pipeline

1. LOD Selection and Streaming.
2. Visibility Buffer Implementation.
3. Deferred Shading Pipeline.
4. Real-time Effects: SSR, SSAO, and MSAA.

### 3.3 Post-Processing

Implementation of bloom, tone mapping, color grading, and Disney Principled BSDF Shader.

## **4 Implementation**

### **4.1 Tools and Libraries**

Describe tools like METIS, QEM, and compute shaders used for implementation.

### **4.2 Algorithms and Data Structures**

Explain key algorithms for LOD management and BVH traversal.

### **4.3 Optimizations**

Detail runtime and memory optimizations employed.

## **5 Results and Analysis**

### **5.1 Visual Output**

Provide rendered scene images and LOD transition comparisons.

### **5.2 Performance Metrics**

Analyze frame rates and memory usage for different scene complexities.

### **5.3 Quality Metrics**

Discuss visual fidelity using SSR, SSAO, and post-processing effects.

## **6 Discussion**

### **6.1 Achievements**

Highlight successful implementations and key contributions.

### **6.2 Limitations**

Discuss challenges and areas needing improvement.

## 6.3 Future Work

Suggest potential enhancements, such as dynamic BVH updates or improved mesh merging algorithms.

## 7 Conclusion

Summarize the objectives, methods, and outcomes, emphasizing the significance of the project.

## 8 Bibliography

### References

- [1] Michael Garland and Paul S. Heckbert. *Surface Simplification Using Quadric Error Metrics*. <https://www.cs.cmu.edu/~./garland/Papers/quadrics.pdf>. Accessed: 2024-11-19. 1997.

## A Appendices

### A.1 Code Samples

Include snippets of key algorithms.

### A.2 Additional Results

Extend performance graphs and rendered images.

### A.3 Glossary

Provide definitions for technical terms used.