MAGMA Module III.1

GEODE

Heterogeneous Labeling Heuristic

Implementation Plan

Michael Salton
Concordia University, Montréal

February 2026

Table of Contents

I. Executive Summary

II. Architecture Overview

III. Pipeline Stages

- Stage 1: Ingestion
- Stage 2: Clustering Engine
- Stage 3: Labeling Engine
- Stage 4: Conversion Engine

IV. Development Phases

- Phase 1: Foundation
- Phase 2: Classification Metrics
- Phase 3: Visualization
- Phase 4: Validation & Tuning
- Phase 5: Conversion & Packaging
- Phase 6: Polish & Documentation

V. Technical Specifications

VI. Output Format (.hga)

VII. Risk Assessment

Appendix A: Tech Stack

Appendix B: Weekly Milestone Checklist

I. Executive Summary

Problem

Hybrid rendering systems that combine meshes and 3D Gaussian Splats currently rely on human labeling or expensive joint-training to decide which representation to use for each region of a scene. This creates a bottleneck for automated pipelines and prevents adoption in real-time applications like video games.

Solution

GEODE is an automated classifier that analyzes pre-trained 3D Gaussian Splatting assets and labels spatial clusters as mesh-optimal or Gaussian-optimal. The classifier uses PCA-based planarity analysis, effective rank distribution, alpha-complexity metrics, and normal coherence to make per-cluster decisions. It operates as a fast preprocessing step at asset import time, requiring no retraining or neural network inference.

Deliverables

- C++ library for 3DGS asset classification
- Clustering engine with spatial hash acceleration
- Labeling engine with configurable thresholds
- Conversion engine for mesh extraction from labeled clusters
- Interactive visualization tool with heat-map display
- CLI interface for batch processing
- Hybrid Gaussian Asset (.hga) packaging format
- Tuned thresholds validated against test datasets
- Documentation and integration guide for MAGMA pipeline

Timeline

14 weeks (3.5 months), broken into 6 phases: Foundation, Classification Metrics, Visualization, Validation & Tuning, Conversion & Packaging, and Polish.

Key Innovation

Unlike existing approaches that treat representation selection as a training-time decision or require joint optimization, GEODE operates as a fast preprocessing step that classifies arbitrary 3DGS assets without retraining. The system takes pre-trained Gaussian assets as input and produces a unified hybrid asset file ready for real-time rendering in game engines.

II. Architecture Overview

Core Problem

Given a pre-trained 3D Gaussian Splatting asset, partition it into spatial clusters and classify each cluster as mesh-optimal or Gaussian-optimal, then convert mesh-optimal regions to triangle meshes and package both representations into a unified hybrid asset.

Design Principle

The classifier operates exclusively on pre-trained 3DGS .ply files. This constraint ensures ground-truth covariance matrices, opacity values, and spherical harmonic coefficients are available for analysis. Mesh and point cloud inputs are explicitly out of scope because computing covariance from local neighborhoods introduces approximation noise that degrades classifier accuracy.

Pipeline Flow

End-to-End Pipeline

Load 3DGS .ply → Ingestion (reconstruct covariance, derive normals) → Clustering Engine (spatial partitioning) → Labeling Engine (per-cluster metrics + classification) → Conversion Engine (mesh extraction + partitioning) → .hga Packaging

Target Use Case

The primary use case is video game asset pipelines. An artist or capture system produces a 3DGS .ply file from photogrammetry or neural reconstruction. GEODE runs at import time (sub-second for typical game assets), producing a .hga hybrid asset that the game engine renders via a multi-pass mesh + Gaussian splatting approach.

III. Pipeline Stages

Stage 1: Ingestion

Purpose: Parse the 3DGS .ply file and construct internal Primitive records with reconstructed covariance, derived normals, eigenvalues, and effective rank.

Process

- Parse the binary .ply file, extracting per-Gaussian position, scale, rotation, opacity, and SH coefficients.
- For each Gaussian, reconstruct the 3×3 covariance matrix Σ from scale (s) and rotation (q) parameters: $R = \text{quat\_to\_mat}(q)$, $S = \text{diag}(s)$, $\Sigma = R \cdot S \cdot S^T \cdot R^T$.
- Compute eigendecomposition of Σ to get eigenvalues $\lambda_1 \geq \lambda_2 \geq \lambda_3$ and derive the normal vector (smallest eigenvector).
- Compute per-Gaussian effective rank: $\text{erank} = \exp(H(q_1, q_2, q_3))$ where $q_i = \lambda_i / \Sigma\lambda_i$.
- Compute scene-level metadata: axis-aligned bounding box, primitive count, centroid, spatial extent.

Output

A flat array of Primitive structs plus scene metadata. Each Primitive contains: position, covariance, eigenvalues (λ_1, λ_2, λ_3), normal, opacity, effective rank, and the original SH DC term.

Stage 2: Clustering Engine

Purpose: Partition the Primitives into spatially coherent clusters that serve as the unit of analysis for classification.

Process

- Step 1: Compute scene bounds. Find the axis-aligned bounding box and spatial extent from all Primitive positions.
- Step 2: Build spatial hash grid. Cell size = extent_per_axis / grid_resolution (default 64). Each cell stores indices of contained Primitives for O(1) neighbor lookup.
- Step 3: Determine k. Initial cluster count k = total_primitives / target_cluster_size. Clamp to [4, total_primitives/50].
- Step 4: Initialize centroids. Use k-means++ initialization: first centroid random, subsequent centroids chosen with probability proportional to squared distance from nearest existing centroid.

- Step 5: Run k-means iterations. Assignment pass: assign each Primitive to nearest centroid. Update pass: recompute centroids as mean of assigned positions. Repeat until convergence or max iterations.
- Step 6: Identify underweight clusters. Flag clusters below minimum threshold (target_size / 5).
- Step 7: Merge underweight clusters. Transfer Primitives to nearest non-flagged cluster by centroid distance.
- Step 8: Identify overweight clusters. Flag clusters above maximum threshold (target_size × 3).
- Step 9: Split overweight clusters. Run k-means locally with k=2 on flagged clusters.
- Step 10: Compute per-cluster metadata. Store cluster ID, primitive indices, bounding box, centroid, primitive count.

Output

An array of Cluster structs ready for the labeling engine.

Stage 3: Labeling Engine

Purpose: Analyze each cluster's geometric and appearance properties and assign a representation label with a confidence score.

Process

- Step 1: Iterate over clusters. Each cluster is processed independently (parallelizable).
- Step 2: Gather Primitives. Collect positions, eigenvalues, normals, opacities, and effective ranks for the cluster.
- Step 3: Compute PCA planarity. Build position covariance matrix, eigendecompose, compute $P = (\Lambda_2 - \Lambda_3) / \Lambda_1$.
- Step 4: Compute effective rank distribution. Compute erank_mean and erank_variance across all Primitives in cluster.
- Step 5: Compute alpha-complexity. Compute α_mean, α_variance, α_min, α_max from opacity values.
- Step 6: Compute normal coherence. Compute mean normal $\bar{n} = \text{normalize}(\Sigma n_i)$, then $C = (1/N) \Sigma |n_i \cdot \bar{n}|$.
- Step 7: Package metrics. Store all computed values in ClusterMetrics struct.
- Step 8: Load thresholds. Read threshold configuration from JSON config file.
- Step 9: Evaluate decision function. MESH_OPTIMAL if: $P > 0.7$ AND erank_mean $\in [1.8, 2.2]$ AND α_mean > 0.85 AND α_var < 0.1 AND $C > 0.7$. Otherwise GAUSSIAN_OPTIMAL.

- Step 10: Check for uncertainty. If any metric is within 10% of threshold, label as UNCERTAIN.
- Step 11: Compute confidence score. Minimum of per-metric normalized distances from thresholds.
- Step 12: Package classification result. Store label, confidence, metrics, cluster ID, primitive indices.
- Step 13: Generate manifest. Assemble full JSON manifest with summary statistics and per-cluster results.

Output

Classification manifest (JSON) containing labeled clusters with metrics, confidence scores, and primitive indices.

Stage 4: Conversion Engine

Purpose: Convert MESH_OPTIMAL clusters to triangle meshes, partition Gaussians by label, and package into unified hybrid asset.

Process

- Step 1: Load manifest and source data. Parse classification JSON and load original .ply.
- Step 2: Partition Gaussians. Separate Primitives into MESH_OPTIMAL, GAUSSIAN_OPTIMAL, and UNCERTAIN subsets by index.
- Step 3: Process MESH_OPTIMAL clusters independently. Each cluster becomes a separate mesh extraction job.
- Step 4: Prepare oriented point cloud. Collect positions and normals, ensure consistent normal orientation.
- Step 5: Run Poisson reconstruction. Feed oriented point cloud into Screened Poisson Surface Reconstruction (depth 8 default). Trim at opacity threshold.
- Step 6: Clean up mesh. Decimate, remove degenerate triangles, remove small disconnected components.
- Step 7: Transfer color. Interpolate SH DC terms from k-nearest Gaussians to mesh vertices using inverse-distance weighting.
- Step 8: Merge cluster meshes. Combine per-cluster patches into single mesh asset.
- Step 9: Handle UNCERTAIN clusters. Default fallback: route to GAUSSIAN_OPTIMAL set.
- Step 10: Write GAUSSIAN_OPTIMAL .ply. Export remaining Gaussians with all original fields preserved.
- Step 11: Write mesh asset. Export merged mesh as .obj with vertex colors.

- Step 12: Package as .hga. Combine mesh, Gaussians, and metadata into binary container format.

Output

A single .hga (Hybrid Gaussian Asset) file containing mesh geometry, Gaussian splat data, and metadata.

IV. Development Phases

Phase 1: Foundation (Weeks 1–3)

Goal: Load 3DGS .ply files, reconstruct covariance, implement spatial clustering.

Week 1: Project Setup + PLY Loader

- Set up CMake project with dependencies (Eigen, tinyply/happly, nlohmann/json)
- Create project structure: src/, include/, tests/, data/
- Implement PLY loader for 3DGS files
  - Parse position, scale, rotation, opacity, SH coefficients
  - Handle binary PLY format from gsplat/nerfstudio outputs
- Write unit tests for loader with sample 3DGS files

Week 2: Primitive Representation + Ingestion

- Design Primitive struct with all required fields
- Implement covariance reconstruction from scale + rotation
- Implement eigendecomposition and normal derivation
- Implement effective rank computation per Gaussian
- Implement Scene class with metadata (bounds, counts)

Week 3: Clustering Engine

- Implement spatial hash grid for O(1) neighbor lookup
- Implement k-means++ initialization
- Implement k-means iteration loop with convergence check
- Implement cluster merge/split for balancing
- Add visualization: export colored point cloud per cluster

Phase 2: Classification Metrics (Weeks 4–6)

Goal: Implement all geometric analysis and classification logic.

Week 4: PCA Analysis + Effective Rank

- Compute position covariance matrix per cluster
- Implement eigenvalue decomposition using Eigen::SelfAdjointEigenSolver
- Compute planarity, linearity, sphericity, surface variation
- Aggregate per-Gaussian effective rank into cluster-level statistics
- Write unit tests with synthetic geometric shapes

Week 5: Alpha-Complexity + Normal Coherence

- Compute alpha statistics: mean, variance, min, max
- Compute normal coherence with absolute dot product
- Validate metrics on known scenes (flat wall, foliage, smoke)

Week 6: Classification Logic

- Implement threshold configuration loading from JSON
- Implement decision function with conjunction of conditions
- Implement UNCERTAIN detection (within 10% of boundary)
- Implement confidence score computation
- Generate classification manifest output

Phase 3: Visualization (Weeks 7–9)

Goal: Build interactive viewer for debugging and inspection.

Week 7: Basic Viewer

- Integrate Polyscope visualization library
- Display scene as point cloud with SH DC colors
- Test with large scenes (500K+ primitives)

Week 8: Classification Visualization

- Implement color mapping: blue (mesh), orange (Gaussian), gray (uncertain)
- Implement confidence gradient (saturated = high, pale = low)
- Implement per-metric heat-map modes
- Add ImGui panel for visualization controls

Week 9: Inspection Tools

- Implement cluster selection via mouse ray-cast
- Create cluster info panel showing all metrics
- Implement threshold adjustment sliders with live re-classification
- Add export: PNG screenshot, JSON manifest, colored PLY

Phase 4: Validation & Tuning (Weeks 10–11)

Goal: Test on real data, tune thresholds, measure accuracy.

Week 10: Test Dataset + Ground Truth

- Collect test assets: Mip-NeRF 360, archaeological scans, synthetic scenes
- Create ground truth labels for 20–30 individual assets
- Implement evaluation metrics: precision, recall, F1 per class
- Establish baseline accuracy

Week 11: Threshold Tuning + Ablations

- Grid search over threshold space
- Ablation study: test classifier with metric subsets
- Document optimal thresholds and metric contributions
- Analyze failure cases

Phase 5: Conversion & Packaging (Weeks 12–13)

Goal: Implement mesh extraction and .hga format.

Week 12: Mesh Extraction

- Integrate Poisson reconstruction (CGAL or Open3D)
- Implement per-cluster mesh extraction pipeline
- Implement mesh cleanup: decimation, degenerate removal
- Implement vertex color transfer from SH DC

Week 13: HGA Packaging

- Design .hga binary format specification
- Implement header, metadata, mesh, Gaussian, cluster map chunks
- Implement .hga writer and reader
- Add optional chunk compression (gzip or Draco)

Phase 6: Polish & Documentation (Week 14)

Goal: Clean up, document, prepare for release.

Week 14: Final Polish

- CLI interface: geode input.ply -o output.hga
- Code cleanup, consistent naming, remove debug prints
- Documentation: README, API docs, algorithm description, integration guide
- Package as library with CMake find_package support

- Memory leak check with Valgrind/ASan

V. Technical Specifications

Core Data Structures

```
struct Primitive {
    vec3 position;
    mat3 covariance;
    vec3 eigenvalues;        // λ₁, λ₂, λ₃
    float alpha;
    vec3 normal;
    float effective_rank;
    std::array<float, 48> sh_coefficients;
};
```

```
struct Cluster {
    uint32_t id;
    std::vector<uint32_t> primitive_indices;
    BoundingBox bounds;
    vec3 centroid;
    uint32_t primitive_count;
};
```

```
struct ClusterMetrics {
    float planarity;
    float linearity;
    float sphericity;
    float surface_variation;
    float erank_mean;
    float erank_variance;
    float alpha_mean;
    float alpha_variance;
    float alpha_min;
    float alpha_max;
    float normal_coherence;
};
```

```
struct ClassificationResult {
    ClusterLabel label;      // MESH_OPTIMAL, GAUSSIAN_OPTIMAL, UNCERTAIN
    float confidence;
    ClusterMetrics metrics;
    uint32_t cluster_id;
    std::vector<uint32_t> primitive_indices;
};
```

Classification Thresholds (Default)

Threshold	Value	Direction
planarity_threshold	0.7	> for mesh
erank_mean_low	1.8	> for mesh
erank_mean_high	2.2	< for mesh
alpha_mean_threshold	0.85	> for mesh
alpha_variance_threshold	0.1	< for mesh
normal_coherence_threshold	0.7	> for mesh

Performance Targets

Stage	Game Asset (50K)	Full Scene (1M)
Ingestion	< 100ms	< 5s
Clustering	< 50ms	< 5s
Classification	< 10ms	< 1s
Conversion	< 500ms	< 30s
Total Pipeline	< 1s	< 45s
Peak Memory	< 200MB	< 2GB

VI. Output Format (.hga)

The Hybrid Gaussian Asset (.hga) format is a binary container that packages mesh geometry, Gaussian splat data, and metadata into a single file for efficient loading by game engines.

File Structure

Chunk	Contents
Header	Magic bytes ("HGA1"), version, file size, chunk count, offset table
Metadata	Asset name, source file, GEODE version, classification config, timestamp, bounding box
Mesh	Vertex positions, normals, colors (packed binary), triangle indices, attribute flags
Gaussian	Positions, scales, rotations, opacities, SH coefficients (packed binary)
Cluster Map	Cluster boundaries, labels, confidences, metrics (for debugging/LOD)
Boundary	Optional: overlap zones, blend weights for mesh-Gaussian transitions

Design Goals

- **Single file per asset:** No separate mesh + Gaussian files to manage.
- **Fast loading:** Chunk offsets enable direct seeks; binary data can DMA to GPU buffers.
- **Optional compression:** Per-chunk gzip or Draco for size reduction; uncompressed for maximum load speed.
- **Self-describing:** Metadata chunk contains everything needed to interpret the file.

VII. Risk Assessment

Risk 1: Thresholds Don’t Generalize

Problem: Optimal thresholds for one scene type may fail on others.

Mitigation: Test on diverse asset types early. Implement scene-type presets (indoor, outdoor, archaeological) if needed.

Fallback: Allow manual threshold tuning via config file.

Risk 2: Clustering Quality Affects Classification

Problem: Poor cluster boundaries may split coherent regions or merge disparate ones.

Mitigation: Experiment with normal-weighted clustering distance. Add boundary refinement pass.

Fallback: Allow manual cluster size adjustment.

Risk 3: Mesh Extraction Quality

Problem: Poisson reconstruction may over-smooth or hallucinate geometry.

Mitigation: Tune octree depth and trim threshold per cluster size. MESH_OPTIMAL clusters are pre-filtered for flatness, reducing Poisson failure modes.

Fallback: Fall back to Delaunay triangulation for problematic clusters.

Risk 4: Performance on Large Scenes

Problem: Scenes with 10M+ primitives may exceed performance targets.

Mitigation: Parallelize clustering and classification with OpenMP. Use spatial hashing for O(1) queries.

Fallback: Add optional downsampling for very large scenes.

Risk 5: Ground Truth Labeling is Subjective

Problem: What “should” be mesh vs. Gaussian is sometimes ambiguous.

Mitigation: Focus validation on clearly mesh (walls, floors) and clearly Gaussian (foliage, smoke) regions. Create labeling guidelines.

Fallback: Use reconstruction quality (Chamfer distance, PSNR) as objective proxy for label correctness.

Appendix A: Tech Stack

Language

C++17 (structured bindings, std::optional, std::filesystem)

Build System

CMake 3.16+

Core Dependencies

- **Eigen 3.4+** — Linear algebra, eigenvalue decomposition (header-only)
- **tinyply or happly** — PLY loading (header-only)
- **nlohmann/json** — Config file parsing (header-only)
- **CGAL or Open3D** — Poisson reconstruction for mesh extraction

Visualization

- **Polyscope** — Interactive 3D visualization with ImGui

Testing

- **Catch2 or Google Test** — Unit testing framework

Optional

- **OpenMP** — Parallel clustering and metric computation
- **spdlog** — Structured logging
- **Draco** — Mesh compression for .hga format

Recommended Development Environment

- IDE: CLion or VS Code with CMake Tools
- Compiler: GCC 11+ or Clang 14+ (Linux), MSVC 2022 (Windows)
- Debugger: GDB/LLDB with IDE integration
- Profiler: Valgrind (memory), perf (CPU)

Appendix B: Weekly Milestone Checklist

Week 1

- ◻ CMake project builds successfully
- ◻ PLY loader parses 3DGS files correctly
- ◻ Unit tests for loader pass

Week 2

- ◻ Primitive struct defined with all fields
- ◻ Covariance reconstruction from scale + rotation works
- ◻ Eigendecomposition and normal derivation works
- ◻ Effective rank computation works
- ◻ Scene class holds primitives with metadata

Week 3

- ◻ Spatial hash grid implemented
- ◻ K-means++ initialization implemented
- ◻ K-means iteration converges correctly
- ◻ Cluster merge/split produces balanced clusters
- ◻ Colored point cloud export works

Week 4

- ◻ Position covariance matrix computation correct
- ◻ Cluster-level eigenvalue decomposition correct
- ◻ Planarity metric matches synthetic tests
- ◻ Effective rank aggregation works

Week 5

- ◻ Alpha statistics computed correctly
- ◻ Normal coherence computed correctly
- ◻ Metrics validated on known scenes

Week 6

- ◻ Threshold config loading works
- ◻ Decision function implemented
- ◻ UNCERTAIN detection works
- ◻ Confidence scores computed

- ◻ Classification manifest output generated

Week 7

- ◻ Polyscope integrated and compiles
- ◻ Scene displays as point cloud
- ◻ Camera controls work smoothly

Week 8

- ◻ Classification colors display correctly
- ◻ Confidence gradient visible
- ◻ Per-metric heat-maps work
- ◻ UI panel controls visualization

Week 9

- ◻ Cluster selection via ray-cast works
- ◻ Cluster info panel shows all metrics
- ◻ Threshold sliders update live
- ◻ Export to PNG/JSON/PLY works

Week 10

- ◻ Test dataset assembled (20+ assets)
- ◻ Ground truth labels created
- ◻ Evaluation metrics implemented
- ◻ Baseline accuracy measured

Week 11

- ◻ Grid search completed
- ◻ Optimal thresholds documented
- ◻ Ablation study completed
- ◻ Failure cases analyzed

Week 12

- ◻ Poisson reconstruction integrated
- ◻ Per-cluster mesh extraction works
- ◻ Mesh cleanup implemented
- ◻ Vertex color transfer works

Week 13

- ☐ .hga format specification complete
- ☐ .hga writer implemented
- ☐ .hga reader implemented
- ☐ Optional compression works

Week 14

- ☐ CLI interface complete
- ☐ Code cleaned up
- ☐ Documentation written
- ☐ Library packaged
- ☐ No memory leaks

GEODE is Module III.1 of the MAGMA (Mesh-Adaptive Gaussian-Mesh Architecture) thesis project.