System Design Visualization (Gerrard Hall)

Goal: build a research-quality, interactive, cinematic visualization of the Gerrard Hall cluster merge process, *faithful to GTSFM outputs* and *matching Frank's four pillars*.

0. What this system must do (success criteria)

1. **Slab layout:** all clusters appear arranged on a thin plane parallel to the camera (2.5D "sheet of glass" look).
2. **Per-cluster inspection:** user can click any cluster and rotate *that cluster only*.
3. **Animated merges:** children clusters move smoothly toward their parent, fade out, parent fades in — no popping.
4. **Cinematic ending:** after final merge, camera runs a short preset flythrough around merged Gerrard Hall, nerfstudio/Rome-style.

1. Inputs from GTSFM (what data we rely on)

From the **GTSFM pipeline outputs directory** you already have:

A. Reconstructions (point clouds)

Each cluster reconstruction folder contains:

- `points3D.txt` (COLMAP format): XYZ + RGB per point
- optionally `cameras.txt`, `images.txt` for frustums/poses

These are already what your viewer loads.

B. Merge tree / hierarchy

GTSFM computes a **cluster tree** and then merges upward. In code this is represented as:

- a tree of clusters (children → parent)
- a list/sequence of merge operations

In your repo you've hard-coded this structure into the `structure` object in `merge-tree.html`.
Later we can **auto-derive this** from GTSFM's cluster/merge logs, but hard-coding is fine for the demo.

C. Camera poses (optional but useful)

For the final cinematic flythrough we'll want:

- final merged reconstruction center + radius (from points)
- optionally real camera pose set for "true upright view framing"

2. System architecture (modules)

Think of the visualization as 4 cooperating engines.

2.1 Layout Engine (slab / thin-box requirement)

Responsibility: decide *where each cluster lives* in the slab, independent of its internal geometry.

Inputs

- merge tree nodes
- per-cluster centroid + bounding sphere
- parent/child relationships

Outputs

- a stable 2D coordinate `(X,Y)` for each cluster on the slab
- a tiny controlled depth `(Z ≈ constant)` for all clusters

Implementation plan

1. **Compute centroids** for each cluster point cloud:
   - `geom.computeBoundingSphere(); center = sphere.center`
2. **Assign slab positions**:
   - simplest first pass: **tree layout in 2D**
     - leaves spread horizontally
     - parents centered above children
   - then scale positions into screen-space slab bounds.
3. **Enforce constant depth**:
   - set all `clusterGroup.position.z = Z0`
4. **Persist positions across time**:
   - layout computed once at load time
   - merge animation *moves clusters toward parent slab position*, not recomputed each frame.

Reference patterns

- City-scale SfM demos place sub-recons in a shallow depth “presentation layer” for readability. *Rome in a Day* videos show progressive clusters in a coherent layout rather than scattered in full depth. Wikipedia+1
- Progressive / hierarchical SfM pipelines explicitly operate on a cluster tree, which matches your merge-timeline metaphor. ACM Digital Library

2.2 Interaction Engine (grab/rotate clusters)

Responsibility: let users select and rotate *one cluster at a time*.

Inputs

- mouse events
- list of cluster THREE.Groups

Outputs

- selected cluster
- per-cluster rotation deltas

Implementation plan

1. **Wrap each cluster in its own group**
   - `const clusterGroup = new THREE.Group();`
   - `clusterGroup.add(pointCloud);`
2. **Raycast on click**
   - cast ray from camera through mouse → find first intersected clusterGroup.
3. **Selection mode**
   - store `selectedCluster`
   - disable global orbit rotation *only while dragging cluster*
4. **Rotate by drag**
   - on mouse move:
     - `selectedCluster.rotation.y += ΔmouseX * k`
     - `selectedCluster.rotation.x += ΔmouseY * k`
5. **Highlight selection**
   - temporary material tweak or outline glow
   - reset previous selection to normal.

Reference patterns

- Interactive SfM viewers (Photo Tourism / Photosynth-style) rely on selecting/isolating sub-structures for inspection. Wikipedia+1

2.3 Merge Animation Engine (smooth converging merges)

Responsibility: convert discrete merge events into continuous motion.

Inputs

- merge timeline events (child paths → parent path)
- slab positions from Layout Engine

Outputs

- per-frame child positions + opacity
- parent opacity

Implementation plan
For each merge event:

1. **Identify child groups + parent group**
2. **Compute target position**
   - `target = slabPos[parent]`
3. **Animate children toward parent**
   - over duration `T` with easing
   - position interpolation:
     - `childPos(t) = lerp(startPos, target, ease(t/T))`
4. **Cross-fade**
   - children opacity → 0 near end
   - parent opacity ramps from 0 → 1
5. **Commit final state**
   - hide children
   - parent remains visible at target.

Optional "force-feel"

- add tiny curved trajectories instead of straight lerp:
  - `childPos(t) = quadraticBezier(start, mid, target, u)`
  - gives the "pulled together" vibe without a full physics sim.

Reference patterns

- *Rome in a Day* visual storytelling uses progressive reveal + smooth convergence rather than popping. [Wikipedia+1](#)

- Modern cluster-tree SfM papers (e.g., Progressive SfM) emphasize hierarchical, progressive merging that maps perfectly onto your animation sequence. ACM Digital Library
- Cycle-Sync is another recent hierarchical global-pose system whose merge/sync storyboards are useful as motion metaphors.

2.4 Camera Engine (merge phase + cinematic phase)

Responsibility: keep a stable overview during merges, then a preset flythrough.

Inputs

- merged reconstruction bounding sphere (center + radius)
- flythrough duration
- camera path function

Outputs

- camera pose each frame

Implementation plan

1. **Phase 1: overview**
    - orbit controls on
    - autoRotate optional
    - camera target = slab center
2. **Phase 2: cinematic flythrough**
    - disable autoRotate & user orbit
    - compute:
        - `flyCenter = mergedSphere.center`
        - `flyRadius = mergedSphere.radius`
    - define spline / circular trajectory around `flyCenter`
    - on each frame:
        - `camera.position = path(u)`
        - `camera.lookAt(flyCenter)`
3. **End**
    - stop at a “hero shot”
    - re-enable orbit controls.

Reference patterns

- Nerfstudio viewer popularized smooth preset orbits / flythroughs for final scenes. Eth Zurich People

- Rome-style SfM demos also end with cinematic orbits around the full reconstruction. [Wikipedia+1]

3. Where each feature touches the pipeline (mapping)

Feature	Needs pipeline data	Where it comes from
Slab layout	cluster centroids, tree structure	centroids from each `points3D.txt`; tree from your `structure` (later: GTSFM cluster tree logs)
Per-cluster rotation	clusters as separate objects	each reconstruction folder → its own THREE.Group
Animated merges	ordering of merges + parent/child links	your `mergeEvents` timeline (later: GTSFM merge schedule)
Flythrough	final merged geometry center/radius, optional real camera upright pose	merged `points3D.txt`; optional `images.txt`/COLMAP poses

4. Implementation order (so you don't drown)

1. **Lock down orientation + slab layout**
    - upright worldGroup rotation (already done)
    - compute slab positions for each cluster
2. **Per-cluster selection/rotation**
    - raycast + drag rotation + highlight
3. **Replace pop-merges with animated merges**
    - lerp → bezier → crossfade
4. **Add cinematic flythrough**
    - clean camera state machine + spline path

Each step is additive; none should change your point size, density, or framing unless you explicitly say so.

5. Extra papers worth mining (beyond the two Frank named)

These are useful specifically for *visual/animation patterns*, not algorithms:

1. **Progressive Structure from Motion (ECCV 2018)** — hierarchical cluster tree SfM, progressive merges, good inspiration for timeline + motion metaphors. ACM Digital Library
2. **Photo Tourism / Photosynth-style SfM viewers** — early gold standard for point-cloud + camera-frustum storytelling and interactive inspection. Wikipedia+1
3. **Nerfstudio (viewer/visualization)** — clean preset camera paths + UI phase switching. Eth Zurich People
4. **COLMAP visualizations / demos** — show best-practice rendering of cameras + point clouds for clarity (good for your final polish). ACM Digital Library
5. **City-scale SfM demo papers in general** (Rome-lineage) — keep skimming citations forward/backward from Rome in a Day like Frank suggested. Wikipedia+1