Frank’s Vision of Visualization and Animation of GTSFM

A research-informed, interactive, cinematic visualization of the GTSFM cluster-merge process

---

1. Purpose of the Visualization

Frank’s goal is not just to “display reconstructions.”
He wants a **research-quality, theoretically-informed interactive visualization** that reflects:

- the *structure* of the GTSFM pipeline
- the *hierarchical* nature of cluster-based SFM
- the *motion, forces, merging, and alignment* between clusters
- the *intuition* behind how large reconstructions come together

This visualization is ultimately an **educational + research tool**, not a simple viewer.

2. High-Level Concept Summary

Frank’s core vision consists of four pillars:

A. A Thin 3D Slab That Represents the Entire Merge Tree

He described the entire set of clusters as living inside:

> “a plane parallel to the viewing direction… a thin box.”

Meaning:

- the entire merge-tree should appear as a *series of slices*, all aligned to a virtual plane

Merge Tree Hierarchy:

- If we think of GTSFM like building a house out of lego pieces
- We start with the small lego clusters (C_1, C_1_1, C_1_2, C_2, C_3, C_4…)

- Each cluster can be considered a partial reconstruction of the building
- C_1_1 and C_1_2 merge → **C_1**
- C_4_1 and C_4_2 merge → **C_4**
- C_1, C_2, C_3, C_4 merge → **FINAL reconstruction**
- Merge Tree Hierarchy : the structure showing which clusters merge to form the final reconstruction
- this is not to be mistaken for the directory this is just the logical structure for who merges into who
- the visualization will play these merges over time

When we say each cluster sits on thin sheets of glass

- each cluster sits on the thin sheet of glass
- C_1_1 sits on glass sheet A
- C_1_2 sits on glass sheet B
- C_1 sits on glass sheet C
- merged (final) sits on glass sheet D
- so the sheets come together to and sit on top of each other to form the final construction
- so this is why we want the visualization to be like a thin box
- this preserves the hierarchical structure -> keeps the structure visually organized -> helps me understand the final reconstruction

- imagine a rectangular sheet of glass floating in front of the camera
- each cluster exists **within this sheet**, not deep into 3D space

Why this matters

- This provides perceptual clarity
- It makes the tree feel *coherent* rather than scattered
- It matches the “2.5D” style of classic SFM system overview diagrams and Rome in a Day animations

Concrete technical requirement

- Build a *layout algorithm* that assigns each cluster a **2D position** within a 3D plane
- Depth (Z) should remain constant or extremely limited
- Clusters should not drift far in depth as merges happen

B. Each Cluster Should Be Grabbable, Rotatable, Inspectable

Frank wants:

> “With the mouse I can go to any of them and spin it around… I can spin each differently.”

Meaning:

- every cluster must be its own **THREE.Group()**
- selecting a cluster lets you rotate only that cluster
- rotating does *not* affect the global layout -> we want to make sure that we understand how the clusters line up inside the thin slab which we described earlier
    - each cluster sits on the page and they are apart from each other -> the big picture and positioning of everything -> if a user grabs just one cluster and rotates it we dont want all the clusters to rotate and the camera to rotate -> we only want the one cluster to rotate
- cluster rotation is isolated to its local coordinate system
    - we need to imagine each cluster is sitting on a small turntable -> we want to be able to spin that turntable but we arent spinning the whole house, the whole desk, or the whole world
    - each cluster gets its own turn table -> this turn table is what we call the 3 group -> we want each cluster to have its own group -> in order to rotate just that cluster
    - rest of the world does not move

Concrete technical requirements

- Raycasting to detect which cluster is clicked
    - We want the user to be able to click on a cluster and we know which cluster the user clicked on
        - We will do this by listening to click on the canvas
        - Convert the mouse position into 3d ray space
            - The world that the ray travels through —
              the 3D environment where your clusters live —
              is the **3D ray space**.
        - Ask 3js -> which object did this ray hit first
        - This process will help us determine which cluster was clicked on
- A “selection mode” to isolate controls to selected cluster
    - Once we know which cluster was clicked we store it as selected cluster = that group
    - Selection mode means -> if a cluster is selected we rotate that cluster on drag events and apply only to this one cluster

- Per-cluster mini-orbit controls
    - Treat the selected cluster as if it has its own orbit controls
        - On mouse drag -> change its rotation
        - Feels like you are orbiting around the cluster
- custom rotation drag handlers
    - When mouse is down on a selected cluster
        - Track how much the mouse moves in X/Y
        - Turn that movement into the changes in rotation angles

- Visual highlight when a cluster is selected
    - To make it obvious what you are currently holding we mark it
    - Brighten point colors
    - Add a glow -> outline affect
    - Change its opacity or size
        - When the selected cluster changes reset the old cluster to normal
        - Apply a special selected material or color to tweak the new one

This transforms the viewer from passive → exploratory → educational

C. Clusters Should Move Automatically According to Merge Events

Frank said:

> "They move because of these forces wherever I am in the player animation."

He does *not* mean physical simulation literally.
He means: **merges should be animated**.

Interpretation

As the merge timeline progresses:

- clusters should **move smoothly** toward their parent's position
- merging is not a sudden hide/show
- there should be a "pulling together" feeling
- children should visually converge before disappearing
- the merged parent should fade in at the correct moment

This resembles:

- force-directed graph animation

- smooth interpolation over time
- motion along a computed trajectory rather than pops

Concrete technical requirements

- trajectory computation for each merge
- interpolation of cluster positions
- cross-fade of child → parent
- timing curves (ease in/out)
- merge “event engine”

This is where research papers help:
Rome in a Day and Cycle-Sync illustrate hierarchical motions, smooth structural updates, and how to visually convey merging structure.

D. Cinematic, Semi-Automated Camera Behavior

He referenced:

- nerfstudio
- Rome in a Day animations
- camera trajectories
- meaningful motion

What he wants

A visualization with *phases*:

1. Merge Timeline Phase

The camera is stable, showing the entire slab.
It should:

- stay centered
- maintain consistent framing
- not drift unpredictably
- possibly make small adjustments for clarity

2. Cluster Interaction Phase

When the user rotates a cluster:

- the camera should not fight the user
- controls remain intuitive and consistent

- optionally, camera nudges focus toward selected cluster

3. LATER we want to work on this after we get a working visualization with the merging clusters to the final visualization that we are happy with:

Final Cinematic Flythrough:

After the full merge:

- camera transitions onto a spline / Bézier curve
- moves around Gerrard Hall in a polished, movie-like orbit
- ends in a nice resting shot

These are exactly the cinematic techniques used in SFM videos accompanying large-scale reconstruction papers.

Concrete technical requirements

- A camera state machine with 3 modes
- Spline-based interpolation
- Center-of-geometry tracking
- Velocity-smoothing for elegance

3. Why Frank Wants You to Read Papers

Frank is encouraging you to:

Not guess — instead reuse established visualization patterns.

Papers like:

1. Building Rome in a Day

Provides insights on:

- camera planning
- progressive reveal of reconstructions
- hierarchical visual storytelling
- cluster merging in animations
- layout decisions

2. Cycle-Sync (Robust Global Camera Pose Estimation)

Helpful for understanding:

- hierarchical cluster merges
- synchronization visualization
- structure-aware animations

Why papers matter to implementation

They contain:

- visual metaphors
- animation sequences
- layout ideas
- camera paths
- diagrams
- transitions

You’re *not* expected to implement their SFM algorithms —
you’re expected to steal their **visual storytelling methods**.

4. Mapping Frank’s Vision → Precise Technical Features

Below is a clean table translating his ideas into buildable components.

Frank’s Statement	Interpreted Requirement	Implementation Feature
“Thin box parallel to viewing direction”	Layout contained within a 2.5D slab	Slab layout algorithm (fixed Z, variable XY)
“Spin each cluster independently”	Per-cluster manipulation	Raycasting, per-group rotation controls
“They move because of these forces”	Animated merges, not pops	Position interpolation + cross-fades
“Look into papers”	Follow established viz patterns	Extract camera paths + transitions
“Camera trajectory like nerfstudio”	Cinematic final flythrough	Spline camera path + timed movement

5. Functional Summary of the Visualization System

Based on Frank's direction, the system needs:

A. Layout System

- compute cluster centroids
- position clusters in a plane
- maintain positions between events
- update smoothly during merges

B. Interaction System

- cluster selection
- rotation-only manipulation
- highlighting
- reset controls

C. Merge Animation System

- schedule event-based transitions
- interpolate positions
- fade children → parent
- keep the slab layout consistent

D. Camera System

- overview camera (merge phase)
- isolated-interaction camera (cluster inspection)
- cinematic flythrough (final phase)

E. Rendering System

- faithful RGB from COLMAP
- density-preserving point size
- stable orientation (upright)

---

6. What "Success" Looks Like

The final visualization should:

- look like a **research paper figure that moves**
- allow users to interact with clusters naturally
- reveal the structure of the GTSFM merge-tree
- follow a clear visual grammar borrowed from major SFM papers
- end with a cinematic sequence around the full reconstruction

If the viewer feels like something that could accompany *Building Rome in a Day*, you've done it correctly.

---

7. Next Steps

Once this document is finalized, we will move to:

1. **Doc 2 — Overview of GTSFM Pipeline**
2. **Doc 3 — System Design (bringing vision + pipeline together)**
3. Begin implementation in your Gerrard Hall repo.