System Design of Animation + Visualization (Gerrard Hall)

Expanded with deep technical/paper-based context for every feature

---

0. Why We Use Research Papers (Frank's Philosophy)

Frank is not asking you to implement new SFM algorithms — GTSFM already does that.
He wants you to study papers because:

These papers solved the *visualization* and *animation* storytelling problems before.

They solved exactly the same problems you are solving:

- How to arrange clusters in a clear way.
- How to display partial reconstructions.
- How to animate merges so they feel natural.
- How to show large-scale reconstructions in a compelling, understandable sequence.
- How to design camera motion that feels meaningful and cinematic.

Your task is to **extract these visual principles** and apply them to the Gerrard Hall viewer.

---

1. Core Feature 1 — Slab Layout (Thin 2.5D Box)

Paper Inspiration: *Building Rome in a Day* (Snavely et al.)

What the Rome paper actually does

In the Rome system, the reconstruction process happens over many partial reconstructions ("clusters"), each built from small subsets of images.
In the supplementary materials + the talks given by Snavely & Seitz, you see:

1. All clusters are visually placed in a coherent "presentation layer," not scattered in 3D space.

Even though the true geometry is 3D and irregular, they place clusters:

- on a shallow depth layer
- arranged spatially in a clean, readable way
- positioned to communicate *relationships*, not actual metric positions

2. They show merge progressions as slices, turning the 3D problem into a visual 2.5D storyboard.

This is exactly what Frank meant by:

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

3. Cluster centroids + a 2D tree layout determine where each cluster appears.

Rome explicitly computes clusters, then overlays them on a clean 2D layout for visualization purposes.
This gives you a very clear precedent:
your slab is conceptually identical to what Rome uses to show hierarchical reconstructions at scale.

How This Informs Your Implementation

A. Compute centroids (same as Rome)

```
geom.computeBoundingSphere();
c_i = boundingSphere.center;
r_i = boundingSphere.radius;
```

B. Compute tree layout (Rome → hierarchical layout)

Using a tidy tree layout (Reingold–Tilford algorithm) gives a stable 2D layout where:

- leaves spread horizontally
- parents centered between children
- depth maps to vertical spacing

This is *exactly* how Rome visualizes incremental merges in the talks/presentations.

C. Enforce constant shallow Z (Rome → presentation layer)

This is the “thin box.”

D. Keep layout static across time (Rome → stable semantics)

Rome does *not* recompute cluster positions every iteration.
They freeze layout once, then animate merges within it.
You should do the same.

---

2. Core Feature 2 — Grabbable, Rotatable, Inspectable Clusters

Paper Inspiration: *Photo Tourism* & *Photosynth* (Snavely et al.)

These systems pioneered:

- selecting a small subset of the reconstruction
- isolating that part
- rotating/orbiting around it
- allowing the user to inspect local geometry
  without disturbing the entire scene.

This is the **direct historical precedent** for Frank’s desire:

> “I want to go to any cluster and spin it around… independently.”

What the Photo Tourism system actually does

In Photo Tourism:

1. Users can **click** on a “sub-reconstruction region.”
2. The viewer isolates that region and creates **local orbit controls**.
3. The global scene remains static in the background.
4. Visual highlighting emphasizes the object of focus.
5. The camera *does not* move the world; it orbits a local center for that substructure.

This maps **identically** to Frank’s requirements.

How This Informs Your Implementation

A. Each cluster must be its own THREE.Group()

This isolates transforms:

```
clusterGroup = new THREE.Group();
clusterGroup.add(pointCloud);
```

B. Raycasting (Photo Tourism → picking substructures)

You detect which cluster the user clicked.

C. “Local orbit” rotation (Photosynth → local turret rotation)

You rotate the cluster’s local frame:

```
clusterGroup.rotation.x += dy * k
clusterGroup.rotation.y += dx * k
```

This is exactly how Photo Tourism rotates individual camera sets.

D. Local highlighting (Photosynth → selected region glow)

Change point colors / opacity.

E. Do NOT move camera for this mode

The camera stays stable, like Photo Tourism keeping global context.

---

3. Core Feature 3 — Animated Merges (Smooth Convergence)

Papers Inspiration:

- **Progressive Structure from Motion (ECCV 2018)**
- **Cycle-Sync (2024)**
- **Rome in a Day (cluster merging animations)**

✔ What these papers do that's relevant

1. They treat merges as a hierarchical progression, not jumps.

Progressive SfM + Cycle Sync describe hierarchical cluster trees, where clusters are merged in stages.

Rome's visualizations show:

- partial reconstructions emerging
- being pulled together
- forming larger aggregates

They do this with *smooth transitions*, not sudden pops.

---

2. Many papers use interpolation to show pose synchronization over time.

Cycle-Sync visualizes synchronization of poses and cluster alignment with:

- smooth position updates
- soft convergence

- graph-like transitions

This gives a perfect metaphor for your merge events.

How This Informs Your Implementation

A. Every merge event becomes a timed animation

Just like Progressive SfM's "progressive updates."

B. Children move toward parent position using easing curves

Inspired by hierarchical motion in Cycle-Sync:

```
childPos(t) = lerp(start, target, easeInOutCubic(u))
```

C. Use a quadratic Bézier arc for "force-directed feel" (Rome uses curved motion)

Rome's smooth cluster visualizations often use curved paths to avoid jitter.

```
pos = quadraticBezier(start, mid, target, e);
```

D. Cross-fade children out & fade parent in

This is how Progressive SfM visualizes the disappearance of outdated partial models.

E. Keep slab layout stable (paper style: stable graph layout)

Rome & Progressive SfM *never* let the layout drift; only substructures move.

---

4. Core Feature 4 — Cinematic Flythrough (Nerfstudio / Rome)

Paper Inspiration:

- **Nerfstudio Viewer (ETH Zürich)**
- **Building Rome in a Day visualizations**
- **COLMAP demos**

These systems produce high-end camera trajectories:

- smooth splines
- constant look-at target
- easing functions
- start wide → approach the scene → orbit → exit on a hero shot

---

What Nerfstudio does technically

1. Define a smooth camera spline

Usually a Catmull-Rom or Bézier curve.

2. Fix a single look-at target

(e.g., the reconstruction center)

3. Move camera position along spline

```
camera.position = spline.getPoint(u);
camera.lookAt(center);
```

4. Use easing for dramatic pacing

easeInOutCubic or logistic curve.

What Rome does

In the Rome talks:

- They show wide establishing shots
- then orbit the major structures
- then zoom into the final merged model
- camera motion is always smooth, preset, non-interactive

This is the exact effect Frank wants.

How This Informs Your Implementation

A. Compute merged bounding sphere

```
center = mergedSphere.center
radius = mergedSphere.radius
```

B. Define orbit path

A simple orbit:

```
theta = u * 2π * 0.5
```

Or a 3-point spline:

```
v1, v2, v3 --> CatmullRomCurve3
```

C. Apply easing

```
u = easeInOutCubic((t-start)/duration)
```

D. Drive in cameraMode === "FLYTHROUGH"

---

5. System Architecture (with paper references attached)

A. Layout Engine

- Based on Rome's hierarchical slabs
- Uses tidy tree layout (common in hierarchical SfM visuals)
- Constant Z ("thin slab" as described by Frank)

B. Interaction Engine

- Based on Photo Tourism's cluster selection
- Local orbit control per cluster
- Visual highlighting (Photosynth)

C. Merge Animation Engine

- Based on Progressive SfM + Cycle-Sync merging
- Smooth bezier trajectories (Rome-like cluster movement)
- Crossfading to express structural transitions

D. Camera Engine

- Nerfstudio spline path techniques
- Rome establishing → orbit → hero shot
- COLMAP-like clean look-at behavior

---

6. Implementation Roadmap (paper-guided)

Step 1 — Slab layout (Rome-inspired)

- build tidy tree layout
- assign fixed Z
- verify stability across merges

Step 2 — Cluster inspection (Photo Tourism-inspired)

- raycast selection
- local rotation controls
- highlighting

Step 3 — Animated merges (Progressive SfM + Cycle Sync)

- implement per-event ribbons
- bezier curves
- fade-out/fade-in

Step 4 — Cinematic Flythrough (Nerfstudio + Rome)

- compute center/radius
- spline path
- easing