VGGT-Ω

Supplementary Material

In this Supplement, we provide the following:

- Additional implementation details, including training configurations, pair construction, the VLM prompt, and annotation filtering criteria in Sec. A.
- Additional discussion in Sec. B.
- Limitations in Sec. C.

A. Additional Details

A.1. Training and Architecture

We set the loss weights to $\lambda_{\text{cam}} = 5.0$, $\lambda_{\text{depth}} = 1.0$, $\lambda_{\text{point}} = 0.5$, and $\lambda_{\text{match}} = 0.1$ to supervise the model. For each scene, following [1, 2], we normalize the ground truth to a unit space. Concretely, we first transform all quantities into the coordinate frame of the first camera and compute the average distance of all 3D points to the origin, then scale the depth maps and translation vectors by this value. As in [1], this normalization is applied only to the ground truth and not to the predictions. To stabilize training, we apply gradient-norm clipping with a threshold of 1.0 and use QKNorm inside the attention layers. At each iteration, we randomly sample the number of input frames from the range $[1, 24]$ and correspondingly set the batch size to saturate GPU memory. Each frame is independently masked by a rectangle whose height and width are uniformly sampled from $[32, 128]$ pixels, with a probability of 0.05. Pixels inside the masked region are set to black, and the corresponding depth values are marked as invalid. For color jittering, we use PyTorch's implementation with brightness 0.5, contrast 0.5, saturation 0.5, and hue 0.1. During self-supervised training, the weights of the teacher model are updated using exponential moving average with decay $\theta = 0.999$. Our camera head follows the implementation in [1], using a ReLU activation for the focal length and no activation for the quaternion and translation parameters.

A.2. Constructing Positive and Negative Pairs

Given per-pixel 3D points obtained by unprojecting the depth maps, we first sample valid pixels in the query frame and assign each sampled pixel to a query patch. Using the known camera intrinsics and extrinsics, we then project the corresponding 3D points into all other frames and retain only those projections that (i) fall inside the image and (ii) are depth-consistent with the target-frame depth maps (within a small relative tolerance of 1% and excluding a narrow image boundary of 4 pixels).

For each target frame, we count how many 3D points from each query patch land in each target patch. This yields, for every query patch, a soft correspondence map over target patches in the form of projection overlap ratios. We select query patches that have sufficient valid projections across views and randomly sample up to a fixed number of them, with sampling probabilities proportional to their total number of correspondences. For each selected query patch, all target patches with $>10\%$ overlap are used to form positive token pairs.

Because some sequences are dynamic, failing the positive-pair criteria does not automatically define a valid negative. Instead, we construct negative pairs by randomly sampling patches as query and target candidates, and then enforcing two constraints: (i) a geometric constraint, requiring the candidate target patch center to lie sufficiently far from the epipolar line induced by the query patch (*i.e.*, to have a large epipolar/Sampson distance), and (ii) an appearance constraint, requiring a sufficiently large ℓ_2 distance between the mean RGB values of the two patches. Patches that satisfy both constraints are treated as negatives. We then subsample these negatives to obtain a balanced set of positive and negative patch pairs for supervising the matching loss on the last-layer tokens.

A.3. VLM Prompt

We use a VLM to discard videos not suitable for multi-view geometry. We prompt the VLM as follows:

```
"You are an expert in computer vision,
photogrammetry, and Multi-View Geometry (MVG)."
"Analyze the video clip to determine if it is
suitable for high-quality 3D reconstruction."
"You must categorize the video and extract scene
metadata based on the strict criteria below."

STEP 1:  CHECK FOR HARD REJECTION FACTORS
``Assess the following.  If ANY are present, the
Classification is 'REJECT_HARD''':
1. Discontinuities & Editing:  Does the video
   contain cuts, dissolves, wipes, fades, or
   montage editing?  (Must be a single continuous
   shot).
2. Non-Physical/2D Content:  Is this a screen
   recording, a slideshow of static images,
   animation, or a cartoon?  (Must be real-world
   footage).
3. Extreme Visual Failure:  Severe motion blur
   (edges lost), severe rolling shutter
   (wobble/jello effect), or
   corruption/glitching.
4. Major Obstructions:  Are there heavy overlays
   (large text/UI), watermarks, or physical
   obstructions like a finger covering the lens?
5. Non-Pinhole Projections:  Is the footage 360°
   equirectangular or heavily distorted fisheye
   without calibration?

STEP 2:  CHECK FOR GEOMETRIC & TEXTURAL
```

```
SUITABILITY
"If the video passes Step 1, assess for
reconstruction quality.  If ANY are present, the
Classification is 'REJECT_SOFT':"
1. Insufficient Parallax:  Is the camera
   stationary?  Does it only rotate (pan/tilt) or
   zoom without physical translation?
   (Translation is required for depth).
2. Texture Issues:  Does the scene lack
   non-repetitive texture?  (e.g., blank white
   walls, clear blue sky only, dark shadows, or
   highly repetitive patterns like grid tiles).
3. Specularities & Transparencies:  Are the
   dominant features reflective (mirrors, water,
   glass) or transparent?  (These create 'ghost'
   geometry).
4. Focus & DOF Issues:  Is there severe focus
   hunting (pulsing), or is the depth-of-field so
   shallow that the background is entirely
   blurred (bokeh)?
5. Dynamic Dominance:  Is the frame dominated
   (>90%) by a moving subject (e.g., a close-up
   talking head) while the static background is
   visible but minimal?

STEP 3:  EXTRACT METADATA
"Determine the scene dynamics:"
1. Dynamic:  Moving objects are present (people
   walking, cars moving, wind blowing trees
   heavily).
2. Static:  The scene is rigid; only the camera
   is moving.

OUTPUT INSTRUCTIONS:
"Return a JSON object strictly following this
schema.  Do not output markdown formatting or
explanations outside the JSON."

{
"classification":
"ACCEPT" | "REJECT_HARD" | "REJECT_SOFT",
"reason": "Brief descriptions of the primary
flaw or 'Good candidate'",
"scene_dynamics": "Static" | "Dynamic"
}

"Label Definitions:"
REJECT_HARD: Unusable due to editing, synthetic
content, severe artifacts, or obstructions.
REJECT_SOFT: Technically usable but risks failure
due to rotation-only, reflections, lack of
texture, or focus issues.
ACCEPT: High-quality candidate.  Single shot,
clear translation/parallax, good texture, rigid
geometry.
```

A.4. Annotation Filtering Criteria

To train the ensemble classifier (XGBoost, Random Forest, and CatBoost) described in Section 3.5.2, we extract several geometric features from each reconstruction. These features are designed to detect common failure modes such as collinear motion degeneracies, inconsistent scale, and noisy camera trajectories. Below we provide the definitions for the important features implemented in our pipeline.

Trajectory Smoothness We quantify the smoothness of the estimated camera trajectory using the acceleration of the translation vectors $\mathbf{t}$. The smoothness score S_{trans} is calculated as the mean squared magnitude of the acceleration:

$$S_{\text{trans}} = \frac{1}{N-2} \sum_{i=1}^{N-2} \|\mathbf{t}_{i+1} - 2\mathbf{t}_i + \mathbf{t}_{i-1}\|^2 \qquad (1)$$

We compute a similar metric S_{rot} for rotations using the second order difference of the rotation vectors. High values in S_{trans} or S_{rot} indicate jittery trajectories, often associated with poor SfM convergence.

Parallax Angle Analysis. To ensure sufficient baseline for multi-view stereo, we analyze the parallax angles of the reconstructed sparse point cloud. For a subset of sparse points $\mathcal{P}$, we identify the set of cameras $\mathcal{C}_p$ visible to point $p \in \mathcal{P}$. We compute the maximum angle subtended by any pair of cameras $c_j, c_k \in \mathcal{C}_p$ at point p. The final feature is the median of these maximum angles across all sampled points. Low median parallax indicates degenerate rotation-only motion or extreme distance.

Point Cloud PCA Shape (Linearity & Planarity). To detect geometric degeneracies such as "fly-by" straight-line reconstructions (which result in cylindrical ambiguity), we perform Principal Component Analysis (PCA) on the normalized point cloud coordinates. Let $v_1 \geq v_2 \geq v_3$ be the eigenvalues of the point cloud covariance matrix. We define *Linearity* as $(v_1 - v_2)/v_1$, *Planarity* as $(v_2 - v_3)/v_1$, and *Scattering* as v_3/v_1. Reconstructions with excessively high linearity are flagged as linear degeneracies and are typically discarded by the classifier.

Depth Map Completeness. This metric evaluates the density of the dense reconstruction stage. For each frame, we calculate the percentage of pixels containing valid, finite depth values relative to the total image resolution. The final feature is the average completeness across all frames. Extremely low values typically indicate failure in the Multi-View Stereo stage, often caused by textureless surfaces, high specularities, or bad cameras.

Point Cloud Noise Level. We estimate the signal-to-noise ratio using statistical outlier detection. For every point, we compute the average distance to its nearest neighbors. Points are classified as noise if this distance exceeds the global average by more than two standard deviations. The feature is the percentage of points classified as noise, which allows us to detect reconstructions plagued by floating artifacts and outliers.

Please note that this pipeline can only provide depth values for rigid pixels, and hence the metrics above have assumed that dynamic pixels are already masked out. We assume the model can learn to estimate depth for dynamic pixels from synthetic data.

B. Discussions

Prioritizing simplicity. Several architectural modifications could further improve the performance. For example, the camera head in VGGT [1] employs iterative refinement, whereas our model predicts camera parameters in a single pass. Adopting a similar refinement strategy could yield an additional 2%–3% gain in AUC@3° across multiple datasets. However, we deliberately forgo this direction to maintain the simplicity of our overall framework. The dense prediction head could also be enhanced by injecting raw RGB values into the feature tokens. More broadly, there remains considerable room to optimize the design of task-specific heads. In this work, we prioritize the quality of the representations learned by the shared feature backbone (aggregator) over maximizing performance through highly specialized prediction heads. This choice is motivated by our observation that once the backbone is well trained, the prediction heads require only 5–10K iterations of fine-tuning.

Auxiliary Inputs. Theoretically, incorporating auxiliary inputs, such as temporal order, camera parameters, depth maps, or scale factors, can further enhance performance. However, we empirically observe that introducing these priors during pretraining, even when randomly applied or masked across training iterations, is often detrimental to general representation learning. Conversely, our preliminary experiments indicate that providing conditional auxiliary inputs exclusively during the fine-tuning phase is highly effective, improving task-specific performance without compromising the integrity of the learned representations. While a comprehensive exploration is beyond the scope of this paper, we believe this represents a promising direction for future research.

C. Limitations

Data ambiguities. Inconsistencies in ground-truth depth annotations across different datasets can lead to confusing model behaviors. For instance, in synthetic datasets such as Aria Synthetic Environments and HyperSim, outdoor scenes are often rendered as 2D textures applied to windows, meaning the GT depth corresponds to the window surface itself. Conversely, in other datasets, depth values correctly represent the actual physical objects visible through windows. As a result, the network struggles to reliably resolve window regions, oscillating between erroneously projecting distant background elements (e.g., buildings or foliage) onto the plane of the window, and correctly estimating their true depth. Similar phenomena also frequently occur with porous or thin-structured objects, such as slatted railings, chain-link fences, and wire meshes. In these cases, datasets inconsistently behave between treating the structure as a solid planar surface (often using alpha-masked textures in synthetic data) and capturing the disjointed depth of the background visible through its gaps.

Annotation pipeline. With decent engineering designs, our data annotation pipeline can effectively estimate camera parameters and depth values for rigid objects. However, like other methods relying on multi-view consistency, it cannot provide accurate depth for dynamic regions in theory. Consequently, our model must learn dynamic priors from synthetic data, which is comparatively orders of magnitude smaller than the rest of the training set.

Masked sensitive content. Due to privacy and licensing constraints, portions of the training data, such as human faces and trademarks, are masked or blurred. As a result, the model may produce unexpected artifacts or unstable predictions in these regions, occasionally leading to visually distorted or qualitatively unappealing outputs.

Dense prediction heads. As discussed in the paper, we replaced the high-resolution convolutional layers with an MLP and pixel shuffle operators. Quantitatively, this design maintains the same performance. Qualitatively, it performs equally well in high-confidence areas; however, in highly uncertain regions (with very low confidence scores), the model tends to predict blocky structures or piecewise constant regions. By default, these uncertain areas should be naturally filtered out via confidence scores, following standard practice [1, 2]. At the same time, once the model backbone is well-trained, people can easily fine-tune a high-resolution convolutional head (*e.g.*, DPT) in just 2-3 hours.

References

[1] Jianyuan Wang, Minghao Chen, Nikita Karaev, Andrea Vedaldi, Christian Rupprecht, and David Novotny. VGGT: Visual geometry grounded transformer. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2025. 1, 3

[2] Shuzhe Wang, Vincent Leroy, Yohann Cabon, Boris Chidlovskii, and Jerome Revaud. DUSt3R: Geometric 3D vision made easy. In *Proc. CVPR*, 2024. 1, 3