PoseGAM: Robust Unseen Object Pose Estimation via Geometry-Aware Multi-View Reasoning

— Supplementary Material —

A. Overview

In this supplementary material, we first examine the fragility of multi-view foundation models under appearance-inconsistent inputs in Appendix B. We then provide additional details about our constructed dataset in Appendix C. Appendix D presents the implementation details of our network architecture, training procedure, inference pipeline, and evaluation setup. In Appendix E, we provide additional quantitative comparisons with existing methods, including runtime and memory costs, the number of reference images, and performance after applying refinement networks, along with ablation studies of our dataset. Finally, we include additional qualitative comparison results in Appendix F.

B. Fragility of Multi-View Foundation Models to Appearance-Inconsistent Inputs

In Fig. 1, we illustrate the fragility of multi-view foundation models when confronted with appearance-inconsistent inputs. The cropped image originates from the real-world query image, whereas the rendering views are generated from the CAD model in a virtual space. Due to differences in lighting, material properties, and other photometric effects, the object's appearance in the query image differs subtly from that in the rendering images.

Since multi-view foundation models such as VGGT [8] are not trained to consider this appearance gap between real and synthetic domains, they often produce highly inaccurate predictions under such mismatched conditions. In Fig. 1, we show the reconstruction results using only the rendering images and using both the rendering images and the query image. As highlighted by the red bounding boxes, the point cloud projected from the query image using the predicted camera pose deviates substantially from the points projected from the rendering images. These inconsistencies appear as large outliers, indicating that the model fails to correctly estimate the camera pose and is therefore sensitive to appearance-inconsistent multi-frame inputs.

This observation also highlights the necessity and practical value of our appearance-editing data type in the large-scale dataset (see Sec. 4.2 in the main paper), which helps mitigate such synthetic-to-real appearance gaps.

C. Additional Dataset Construction Details

Given access to the object model $\mathcal{M}$, we synthesize additional geometry-related data beyond the texture-rendered

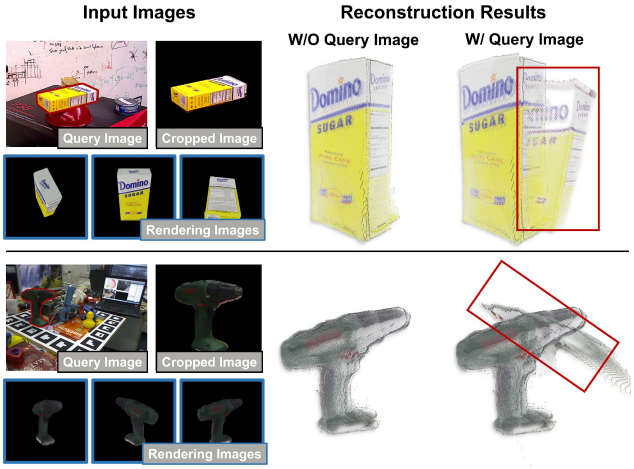

Figure 1. **Analysis of the Fragility of Multi-View Foundation Models to Appearance-Inconsistent Inputs.** The input set consists of several rendered images along with a cropped region from the query image. When only rendered images are provided, VGGT [8] predicts consistent poses across views, as evidenced by the clean and coherent reconstructed point cloud without outliers. These point clouds are obtained by back-projecting image pixels into 3D using the predicted camera poses. However, once the cropped query image is included, VGGT fails to maintain consistency: the predicted pose for the query image becomes inaccurate. The red bounding box highlights the resulting deviations in the reconstructed point cloud, demonstrating the sensitivity of the model to appearance-inconsistent inputs.

image I_i under each manually defined camera pose T_i. Specifically, we first render a depth map D_i at pose T_i using a graphics renderer such as Blender [1] or Nvdiffrast [5]. Since both the camera pose T_i and the intrinsic matrix of the rendering camera are known, we can reconstruct the corresponding point map in world coordinates, denoted as P_i, through the standard camera projection model.

In addition to depth, we also render the normal map O_i and the object mask M_i using the graphics renderer. These geometry-aware data modalities satisfy the input requirements of the PTv3 [9] geometry representation network used in our method (see Appendix D.1).

Fig. 2 presents additional visual examples to illustrate the full composition of our dataset. We also compare the statistics of our dataset with existing datasets in Appendix E, along with an ablation study analyzing the contribution of each subset.

Figure 2. **Examples from the constructed object pose estimation dataset.** From left to right: object mesh after texture rebaking, basecolor rendering image, depth map, normal map, and four types of rendered images used as query inputs during training (see Sec. 4.2 in the main paper for details).

D. Implementation Details

D.1. Network Architecture

Input Data. As illustrated in Fig. 2 of the main paper, the network input consists of two components: the query image I_{query} and multi-view data rendered around the object model $\mathcal{M}$. The multi-view data comprises four modalities: camera parameters $\mathcal{T}$, RGB images $\mathcal{V}$, point maps $\mathcal{P}$, and geometric feature maps $\mathcal{F}$. These modalities are paired such that the elements at the same index correspond to the same viewpoint of the object model. Note that the point maps $\mathcal{P}$ and geometric feature maps $\mathcal{F}$ are not directly available;

they are derived from depth maps and camera intrinsics, and from point maps, as described in Appendix C and detailed further below.

Geometry Feature Extractor. As described in Sec. 3.3 of the main paper, we employ a geometry representation network to extract a global 3D representation of the object. Specifically, we use PTv3 [9]. The original PTv3 takes as input a point cloud $\mathbb{R}^{S\times 9}$, where S denotes the total number of points and each point encodes 3D coordinates, surface normals, and color features.

To allow rearrangement of the extracted global representation into the view-based format, we use point maps $\mathcal{P}$ instead of directly sampling points from the object mesh surface. Given that the camera poses $\mathcal{T}$ are carefully sampled to cover the entire object (see Appendix D.2), the aggregation of point maps across all rendered views sufficiently represents the full object and serves as input to PTv3. Specifically, for each rendered view, we use the object mask to select valid pixels and construct the per-view point cloud:

$$\mathrm{PC}_i = \mathrm{cat}\big[P_i, O_i, I_i\big]_{M_i} \tag{1}$$

Each point in PC_i contains a 3D coordinate, a surface normal, and an RGB color. All per-view point clouds are then concatenated to form the final input point cloud: $\mathrm{PC} = \bigcup_{i=1}^{N} \mathrm{PC}_i$.

After encoding with PTv3, each point is embedded into a feature vector ($\mathbf{e}_i \in \mathbb{R}^C$). To redistribute these features back into a multi-view format, we restore their spatial positions using the mask indices that record each point's source pixel coordinates (u_i, v_i) in the corresponding view. The per-view feature map is reconstructed as:

$$F_i(u,v) = \begin{cases} \mathbf{e}_i, & \text{if } (u,v) = (u_i, v_i) \text{ for point } i, \\ \mathbf{0}, & \text{otherwise.} \end{cases} \tag{2}$$

In practice, since each viewpoint contains a large number of pixels (over half a million), aggregating all views at full resolution would incur excessive computational and memory costs. To address this, we downsample the inputs P_i, O_i, and I_i prior to feature extraction.

Multi-Modal Encoders. The network employs four separate encoders to process each modality. For RGB images, we use DINOv2 [7] as the image encoder, producing feature tokens X. Camera parameters are represented as a 9-dimensional vector, comprising 4 parameters for rotation (quaternion), 3 for translation, and 2 for the field of view. These are encoded into a single camera token $\mathbf{c}$ using a simple MLP. For the point maps and geometric feature maps, we apply a strided convolutional layer followed by flattening, producing patch tokens for each modality.

Fusion Transformer. The tokens from all encoders are processed by a stack of 24 attention blocks, each consisting of both cross-attention and self-attention layers. In the cross-attention layers, the combination of image tokens and camera tokens $(X, \mathbf{c})$ serves as the query. Since the query image I_{query} has no known camera extrinsics, a learnable token is assigned in its place. The point map tokens and geometric feature tokens are used as the key and value tokens in the cross-attention layers.

For the self-attention layers, we adopt the alternating intra-frame and inter-frame self-attention mechanism from VGGT [8]. Intra-frame self-attention computes relationships among tokens within the same viewpoint, while inter-frame self-attention captures correlations across tokens from different viewpoints, modeling multi-view interactions.

Pose Output Decoder. From the output tokens of the Fusion Transformer, we select only those corresponding to the input camera tokens (for the query image, this corresponds to the learnable token). These tokens, one per viewpoint, are passed to a camera head whose architecture is inherited from VGGT [8]. The camera head consists of several self-attention layers followed by an MLP, which decodes each token into a 9-dimensional vector representing the camera extrinsics and intrinsics in the same format described above.

D.2. Model Training, Inference, and Evaluation

Training Details. During training, we freeze the pretrained DINOv2 [7] weights for image feature extraction. For PTv3, we use pretrained weights from Sonata [10] and set its learning rate to 5×10^{-5}. For the self-attention layers and the camera head, pretrained weights from VGGT [8] are used, with the same fine-tuning learning rate of 5×10^{-5}. All other layers use a learning rate of 2×10^{-4}, including the cross-attention layers and the encoders for camera, point maps, and geometric feature maps. The full network contains approximately 1.5B parameters and is trained for 100k iterations on 4 A100 GPUs, taking roughly 8 days.

To improve training efficiency, we adopt dynamic batch training, similar to VGGT [8], where each object is represented by a randomly selected 11–24 multi-view frames. Ten of these camera poses are used as known views, sampled via farthest point sampling (FPS) from the 50 camera poses prepared for each object (see Sec. 4.2 in the main paper) to ensure full coverage. For these known views, we use basecolor-rendered images (see Appendix C) to minimize the influence of environmental lighting. The remaining frames are used as unknown query images, randomly selected from the four types of rendered images (Sec. 4.2 in the main paper). The maximum number of frames per batch is set to 48.

For data augmentation, in addition to standard transfor-

mations such as color jittering and grayscale conversion, we apply random 3D rotations to the input object models. Corresponding rotations are applied to normal maps, point maps, and camera extrinsics to maintain alignment. We also apply 2D image-space random rotations to the RGB inputs, with corresponding adjustments to normal maps, point maps, depth maps, mask maps, and extrinsics for each frame. To simulate practical occlusion scenarios, objects are randomly masked using ellipses, rectangles, and free-form masks [11]. To better model noise introduced by imperfect object segmentation in query images, we randomly overlay or underlay images of other objects onto the query image. Additionally, to mimic blur artifacts introduced during cropping and scaling, we apply JPEG compression and Gaussian blur to the images.

To stabilize training despite varying object sizes, all objects are normalized to fit within a bounding box of $[-B, B]$ along each axis, with $B = 1.05$. The training objective is a camera pose regression loss, formulated as an L1 loss between predicted and ground-truth camera parameters. Both known and query views are supervised, but the loss from known views is weighted by 0.5 to encourage the network to focus on the more challenging query-view estimation. Unlike VGGT [8], which predicts relative poses between frames, our method directly predicts absolute camera poses, allowing direct use of the geometry from the object model $\mathcal{M}$ defined in an absolute coordinate system.

Inference Details. To remain consistent with training in a normalized object space, inference is performed on a normalized version of the object. The predicted pose is then rescaled to account for the normalization, producing the final object pose in the real-world scale.

Specifically, given a query image I_{query} and an object model $\mathcal{M}$, we first normalize $\mathcal{M}$ into the canonical bounding box used during training. Let s denote the isotropic scaling factor that maps points from the original object to the normalized one. The network predicts the camera pose relative to the normalized object, denoted as $T^{\text{Cam}}_{\text{norm}}$. Since the desired output is the object-to-camera transformation, we invert this prediction: $T^{\text{norm}}_{\text{Cam}} = (T^{\text{Cam}}_{\text{norm}})^{-1}$.

To rescale this normalized extrinsic back to the true object scale, consider any 3D point $\mathbf{q}$ in the original object frame. Its normalized counterpart is $\mathbf{q}_{\text{norm}} = s\,\mathbf{q}$. Let the network-predicted camera pose with respect to the normalized object be $T^{\text{norm}}_{\text{Cam}} = [\mathbf{R}_{\text{norm}}, \mathbf{t}_{\text{norm}}]$. We denote perspective projection under camera intrinsics K by (we slightly abuse notation here and let $\mathbf{q}$ denote the transformed 3D point in the camera frame):

$$\pi(\mathbf{q}) = \text{div}(K\mathbf{q}) = \left[\frac{(K\mathbf{q})_x}{(K\mathbf{q})_z}, \frac{(K\mathbf{q})_y}{(K\mathbf{q})_z}\right] \tag{3}$$

where div indicates division by the depth coordinate, and $(\cdot)_x$, $(\cdot)_y$, $(\cdot)_z$ denote the horizontal, vertical, and depth components.

To ensure that the projected 2D pixel location of the object remains unchanged before and after rescaling, we seek a rotation $\mathbf{R}$ and a translation $\mathbf{t}$ such that:

$$\pi(\mathbf{R}_{\text{norm}}(s\,\mathbf{q}) + \mathbf{t}_{\text{norm}}) = \pi(\mathbf{R}\mathbf{q} + \mathbf{t}) \tag{4}$$

Since both sides use the same intrinsics and perspective projection is defined up to an arbitrary nonzero scale before depth division, this is equivalent to enforcing:

$$\frac{\mathbf{R}_{\text{norm}}(s\,\mathbf{q}) + \mathbf{t}_{\text{norm}}}{[\mathbf{R}_{\text{norm}}(s\,\mathbf{q}) + \mathbf{t}_{\text{norm}}]_z} = \frac{\mathbf{R}\mathbf{q} + \mathbf{t}}{[\mathbf{R}\mathbf{q} + \mathbf{t}]_z} \tag{5}$$

Dividing the numerator and denominator on the left-hand side by s gives:

$$\frac{\mathbf{R}_{\text{norm}}\mathbf{q} + \mathbf{t}_{\text{norm}}/s}{[\mathbf{R}_{\text{norm}}\mathbf{q} + \mathbf{t}_{\text{norm}}/s]_z} = \frac{\mathbf{R}\mathbf{q} + \mathbf{t}}{[\mathbf{R}\mathbf{q} + \mathbf{t}]_z} \tag{6}$$

Therefore, the pose estimation for the real object is:

$$T_{\text{query}} = [\mathbf{R},\, \mathbf{t}] = [\mathbf{R}_{\text{norm}},\, \frac{\mathbf{t}_{\text{norm}}}{s}] \tag{7}$$

Evaluation Details. When evaluating our method on BOP benchmark datasets, we have access to the 3D object model, the query image, the camera intrinsics for the query image, and the segmentation (from CNOS [6]) of the target object. We first select 10 camera poses around the 3D object (ensuring full coverage via farthest point sampling), using intrinsics the same as the query image, to render multi-view frames. For the query image, since the object often occupies only a small region, we crop the object using the available segmentation map and resize it to the same resolution as the original query image. We then perform pose estimation following the procedure described in the Inference Details above.

Since the network outputs extrinsics relative to the output intrinsics, which differ from the target intrinsics, we convert the estimated pose $(\mathbf{R}_{\text{src}}, \mathbf{t}_{\text{src}})$ to the target intrinsic space using SVD. Specifically, we map the source projection matrix into the target intrinsic space via $K^{-1}_{\text{dst}} K_{\text{src}} [\mathbf{R}_{\text{src}} \mid \mathbf{t}_{\text{src}}]$, and extract the closest rotation and translation through SVD, yielding $(\mathbf{R}_{\text{dst}}, \mathbf{t}_{\text{dst}})$ whose projection under K_{dst} approximates the original image.

However, since $K^{-1}_{\text{dst}} K_{\text{src}}$ contains non-uniform scalings and translations, the rotation matrix cannot be recovered accurately. Consequently, the SVD-based rotation–translation approximation introduces error, preventing recovery of the extrinsics. To further compensate for this, we leverage an IoU loss and a 2D Chamfer loss to compare the input segmentation map with the rendering mask generated using the SVD-converted extrinsics. The extrinsics are then updated through some iterations of gradient descent. Although this solution depends on the quality of the segmentation map, it generally produces reliable results in our evaluation.

Figure 3. **Visual comparison with other methods.** From left to right: the query image for pose estimation, object masks with each object shown in a distinct color, and the projection results of different methods after applying the estimated poses to the 3D object models. The projected 3D models are outlined with borders matching the colors of their corresponding masks. The last three rows show samples from the T-LESS dataset [3]; since this dataset does not provide object textures, we display the rendered normal images of the objects for clearer visualization.

E. Additional Quantitative Results

Runtime, Memory Cost, and Number of Reference Views. Table 1 presents runtime and memory comparisons across different methods, while Table 2 reports the number of reference views used by each method for Table 1 in the main paper. As shown, our method requires fewer reference views than competing approaches. Benefiting from this design, our method incurs negligible onboarding cost.

Method	Time per Object Detection		Peak GPU Memory
	Onboarding (Offline, e.g., rendering)	Pose Estimation (Inference)	
MegaPose	∼0s	1.47s	3.33GB
GigaPose	1.77s	0.26s	6.47GB
FoundPose	1100.37s	0.46s	0.83GB
Ours	0.30s	0.67s	9.49GB

Table 1. Runtime and peak GPU memory comparisons, measured per object detection.

Method	OSOP	ZS6D	MegaPose	GenFlow
# Ref. Views	>1K	300	576	>100
Method	GigaPose	FoundPose	RayPose	Ours
# Ref. Views	162	798	16 (8 coarse + 8 fine)	**10**

Table 2. Comparison of the number of reference views.

Dataset	# Objects	# Query Images	Object Sources
MegaPose 2M	53k	2M	ShapeNet, Google-Scanned-Objects
Ours	**190k**	**30M**	Objaverse, Toys4K, 3D-FUTURE, ABO, HSSD

Table 3. Comparison of dataset scale and object sources.

MegaPose [4] also achieves low onboarding cost; however, it shifts the computational burden to the inference stage. In terms of memory consumption, our method requires more memory due to its larger network size, but still maintains competitive inference speed.

Dataset Comparisons and Ablations. Table 3 compares our dataset with the synthetic dataset from MegaPose [4]. Our dataset contains a larger number of object assets, more query images spanning diverse scenarios, and a wider range of object data sources.

In Table 4, we analyze the effect of training for 10k iterations under different dataset settings. The results show that our dataset consistently yields better performance than the MegaPose dataset. We further conduct ablations of our dataset, which verify the effectiveness of our data construction across different scenarios.

Comparisons with Refinement Networks. Table 5 presents performance comparisons after applying the MegaPose refinement network [4]. Specifically, we use the pose estimates reported in Table 1 of the main paper for each method as coarse initializations, which are then refined using the refinement network. As shown in the table, our method benefits much from the additional refinement. Owing to its accurate initial pose estimates, our method achieves performance that remains competitive with other approaches.

MegaPose 2M	Our dataset scenarios				AR (LM-O)
	Centric	Uncentric	EnvMap	Edited	
✓	✗	✗	✗	✗	28.5
✗	✓	✓	✓	✓	**34.3**
✓	✓	✓	✓	✓	34.2
—	✓	✗	✗	✗	22.6
	✓	✓	✗	✗	30.8
	✓	✓	✓	✗	33.6
	✓	✓	✓	✓	**34.3**

Table 4. Performance under different training data settings. "Centric", "Uncentric", "EnvMap", and "Edited" correspond to the four components of our dataset described in Sec. 4.2 of the main paper, respectively. Evaluation is conducted on the LM-O [2] subset.

Method	LM-O	T-LESS	TUD-L	IC-BIN	YCB-V	Average
Ours (coarse)	43.0	34.1	56.8	24.3	47.4	41.1
MegaPose	49.9	47.7	65.3	36.7	60.1	51.9
GigaPose	55.6	**54.6**	57.8	**44.3**	63.4	55.1
FoundPose	55.7	51.0	63.3	43.3	**66.1**	55.9
RayPose	56.2	53.8	**66.5**	41.6	62.8	56.2
Ours	**57.4**	52.1	65.7	44.1	63.3	**56.5**

Table 5. Performance after applying refinement network.

F. Additional Qualitative Results

In Fig. 3, we present additional qualitative results, comparing our method with existing approaches.

References

[1] Blender Foundation. Blender, 2024. 1

[2] Eric Brachmann, Alexander Krull, Frank Michel, Stefan Gumhold, Jamie Shotton, and Carsten Rother. Learning 6d object pose estimation using 3d object coordinates. In *ECCV*, 2014. 6

[3] Tomáš Hodaň, Pavel Haluza, Štěpán Obdržálek, Jiří Matas, Manolis Lourakis, and Xenophon Zabulis. T-LESS: An RGB-D dataset for 6D pose estimation of texture-less objects. *WACV*, 2017. 5

[4] Yann Labbé, Lucas Manuelli, Arsalan Mousavian, Stephen Tyree, Stan Birchfield, Jonathan Tremblay, Justin Carpentier, Mathieu Aubry, Dieter Fox, and Josef Sivic. Megapose: 6d pose estimation of novel objects via render & compare. In *CoRL*, 2022. 6

[5] Samuli Laine, Janne Hellsten, Tero Karras, Yeongho Seol, Jaakko Lehtinen, and Timo Aila. Modular primitives for high-performance differentiable rendering. *ACM Transactions on Graphics*, 39(6), 2020. 1

[6] Van Nguyen Nguyen, Thibault Groueix, Georgy Ponimatkin, Vincent Lepetit, and Tomas Hodan. Cnos: A strong baseline for cad-based novel object segmentation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV) Workshops*, pages 2134–2140, 2023. 4

[7] Maxime Oquab, Timothée Darcet, Theo Moutakanni, Huy V. Vo, Marc Szafraniec, Vasil Khalidov, Pierre Fernandez, Daniel Haziza, Francisco Massa, Alaaeldin El-Nouby, Russell Howes, Po-Yao Huang, Hu Xu, Vasu Sharma, Shang-Wen Li, Wojciech Galuba, Mike Rabbat, Mido Assran, Nicolas Ballas, Gabriel Synnaeve, Ishan Misra, Herve Jegou, Julien Mairal, Patrick Labatut, Armand Joulin, and Piotr Bojanowski. Dinov2: Learning robust visual features without supervision, 2023. 3

[8] Jianyuan Wang, Minghao Chen, Nikita Karaev, Andrea Vedaldi, Christian Rupprecht, and David Novotny. Vggt: Visual geometry grounded transformer. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pages 5294–5306, 2025. 1, 3, 4

[9] Xiaoyang Wu, Li Jiang, Peng-Shuai Wang, Zhijian Liu, Xihui Liu, Yu Qiao, Wanli Ouyang, Tong He, and Hengshuang Zhao. Point transformer v3: Simpler, faster, stronger. In *CVPR*, 2024. 1, 3

[10] Xiaoyang Wu, Daniel DeTone, Duncan Frost, Tianwei Shen, Chris Xie, Nan Yang, Jakob Engel, Richard Newcombe, Hengshuang Zhao, and Julian Straub. Sonata: Self-supervised learning of reliable point representations. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pages 22193–22204, 2025. 3

[11] Jiahui Yu, Zhe Lin, Jimei Yang, Xiaohui Shen, Xin Lu, and Thomas S Huang. Free-form image inpainting with gated convolution. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 4471–4480, 2019. 4