6D pose estimation of novel objects using AI generated meshes and FoundationPose

Bachelor's Project Thesis

Iustin Lungu, s4787641, i.lungu.2@student.rug.nl,
Supervisors: prof. Hamidreza Kasaei

Abstract: Robust 6D object pose estimation in real-world settings is often constrained by the availability of high-fidelity computer-aided design (CAD) models. This thesis investigates whether generative artificial intelligence (GenAI) models can bridge this gap by reconstructing viable 3D meshes from single-view RGB (red–green–blue) images. We integrate meshes generated by diffusion-based methods, Magic123, Zero123, and DreamFusion, into a pose estimation pipeline based on FoundationPose, and evaluate their effectiveness on 6D pose accuracy. Experiments are conducted on two fundamentally different datasets: the structured LINEMOD benchmark, which provides known objects with CAD models, and the unstructured HOTS dataset, which features novel, cluttered scenes without any available 3D geometry. Our results show that, despite variability in reconstruction quality, GenAI meshes can support reliable pose estimation under high-fidelity reconstructions, but coarse or noisy outputs lead to large failures, highlighting the need for improved reconstruction fidelity or more robust refinement. These findings support the use of generative models as viable substitutes for CAD assets, advancing scalable 6D pose estimation in unconstrained, real-world domains. The source code for this thesis is publicly available at our GitHub Repository: `https://github.com/JustinLungu/FoundationPose-BachelorThesis/tree/main`.

1 Introduction

1.1 Motivation and Background

Estimating the 6D pose of objects, comprising their 3D position and orientation, is a core capability in robotics and augmented reality, enabling tasks such as object manipulation, scene understanding, and spatial reasoning (Brachmann et al., 2014). Classical pipelines often rely on precise 3D CAD models for accurate pose estimation (Hodan et al., 2018), but these assets are rarely available in dynamic or open-world environments, limiting scalability in practical deployments.

Recent advances in GenAI offer a promising alternative: synthesizing 3D geometry from 2D inputs. Single-view 3D reconstruction using diffusion models has shown the potential to bypass CAD dependency by generating object meshes directly from images or prompts. In this study, we explore three representative models, Zero123 (Gao et al., 2023), Magic123 (Qian et al., 2023), and DreamFusion (Poole et al., 2022), that reflect a spectrum of generative strategies and input modalities. These models serve as replacements for CAD assets, enabling scalable 6D pose estimation in scenarios where ground-truth geometry is unavailable. However, their output quality, particularly in terms of geometric fidelity to real-world shape and scale, varies widely. Their downstream usability in pose estimation tasks remains largely underexplored.

1.2 Applications and Challenges

Robust 6D pose estimation has immediate relevance in robotic manipulation, warehouse automation, and mixed-reality systems (Pöllabauer et al., 2024). Yet, recovering 3D shape from a single RGB view remains an ill-posed problem due to occlusions, depth ambiguities, and texture sparsity (Tatarchenko et al., 2016). Additionally, downstream pose estimators such as FoundationPose (Wen et al., 2024) are typically trained on ideal meshes or clean 3D models, making it unclear how well they generalize when fed reconstructions from generative methods.

Recent evaluations show that pose estimation accuracy can vary significantly in real-world settings due to factors such as material reflectance, clutter, and sensor noise (Pöllabauer et al., 2024). In practice, these variations may lead to critical failures in robotics pipelines, such as inaccurate object alignment or misgrasping, when mesh artifacts or occlusion distort the estimated pose. These practical challenges underscore the need for methods that can generalize beyond controlled datasets and synthetic training conditions.

1.3 Objective and Research Scope

This thesis investigates whether diffusion-based generative models can provide usable 3D object representations in the context of 6D pose estimation. Specifically, we benchmark FoundationPose with GenAI meshes (Magic123, Zero123, DreamFusion) versus CAD models on two datasets:

- **LINEMOD** (Hinterstoisser et al., 2012), a standard 6D pose benchmark with RGB-D (color + depth) data and known CAD geometry.
- **HOTS** (Tziafas, 2023), a real-world robot-vision dataset with RGB and segmentation masks but no available 3D geometry.[*]

This setup lets us evaluate both established accuracy and out-of-domain generalization.

1.4 Research Questions (RQs)

To accomplish these goals, the central questions guiding this research are:

- **RQ1:** How well does FoundationPose generalize to novel, unstructured datasets?
- **RQ2:** How accurately do 3D diffusion models reconstruct object geometry compared to CAD?
- **RQ3:** How robust is 6D pose estimation when using AI-generated meshes or applying controlled noise to CAD?

Through this study, we aim to advance scalable alternatives for 6D pose estimation in settings where 3D assets are not readily available.

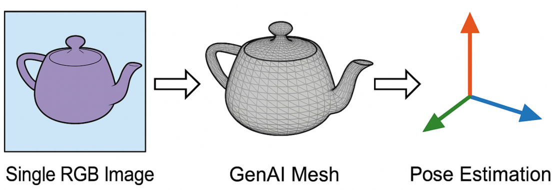


Figure 1.1: Overview of the 6D pose estimation pipeline: a single RGB image is used to generate a 3D mesh via generative AI, which is then processed by FoundationPose to estimate object pose.

2 Related Works

2.1 6D Pose Estimation Overview

6D object pose estimation involves determining an object's position and orientation in 3D space, typically represented by a translation vector and a rotation matrix. It is fundamental for robotic manipulation, augmented reality, and spatial understanding tasks. Traditional methods include geometric algorithms such as Perspective-n-Point (PnP) (Lepetit et al., 2009) and Iterative Closest Point (ICP) (Besl and McKay, 1992), which often require accurate keypoint or depth information. More recent learning-based methods leverage deep neural networks to directly predict poses from RGB or RGB-D input (Brachmann et al., 2014). Benchmarks like LINEMOD have established standard evaluation protocols using metrics such as average distance of model points (ADD) and its symmetric variant ADD-S, as defined in the Benchmark for Object Pose (BOP) protocol (Hodan et al., 2018).[†]

Given the centrality of 6D pose estimation to real-world applications, recent research has shifted toward unified, learning-based pipelines that can generalize across object types and input modalities. Among them, FoundationPose (Wen et al., 2024), a general-purpose framework developed by NVIDIA, stands out as a comprehensive and modular solution. Its flexibility and performance make it a key component of this thesis, particularly for analyzing how the fidelity of 3D object representations influences pose estimation accuracy.

2.2 FoundationPose Overview

FoundationPose supports both model-based and model-free pipelines for 6D pose estimation. The framework is built around two core modules: a Scorer, which evaluates pose hypotheses by comparing rendered views with observed images, and a Refiner, which optimizes pose alignment through image-space feedback. In model-based mode, FoundationPose uses 3D meshes for rendering. In model-free mode, it trains Neural Object Fields (NOFs) from few-shot RGB-D inputs. This dual-mode design makes it well-suited for studying the impact of mesh quality on pose accuracy. However, due to limited documentation on the model-free setup, our experiments focused solely on the model-based mode.

2.3 Traditional 3D Reconstruction vs. Generative AI

Conventional reconstruction pipelines, Structure-from-Motion (SfM) (Schönberger and Frahm, 2016), Multi-View Stereo (MVS) (Seitz et al., 2006), and volumetric fusion from RGB-D captures (Curless and Levoy, 1996) rely on many calibrated viewpoints or depth scans to build accurate geometry. While these methods excel in controlled settings,

[*] The HOTS dataset offers RGB-D observations from a robot-mounted camera in cluttered, occlusion-heavy environments, providing both object- and scene-level views that reflect industrial deployment scenarios.

[†] Another well-known benchmark is T-LESS (Hodan et al., 2017), which focuses on texture-less industrial objects, but it is not used in this study.

Figure 2.1: Examples of 6D pose estimation with FoundationPose. Left: mustard demo; Right: drill scene. Green shows the aligned mesh, and red/green/blue axes indicate the estimated orientation.

they are inherently batch-oriented, computationally heavy, and ill-suited for single-image scenarios.

By contrast, diffusion-based generative models learn 3D shape priors that enable mesh generation from a single RGB input or prompt (Figure 2.2). In this study, we investigate three representative GenAI models that differ in modality, reconstruction strategy, and geometric quality:

- **Zero123** (Gao et al., 2023) is an image-conditioned diffusion model originally designed for novel view synthesis. While not intended for mesh reconstruction, it can be adapted for 3D generation using frameworks such as (Threestudio, 2023), which enable mesh export.

- **Magic123** (Qian et al., 2023) is a hybrid image- and text-conditioned model that produces high-quality 3D meshes using Score Distillation Sampling (SDS) and pretrained diffusion priors. It offers a favorable trade-off between realism and geometric consistency.

- **DreamFusion** (Poole et al., 2022) is a text-to-3D model based on SDS, capable of producing detailed meshes from descriptive prompts. While not image-conditioned by default, it can be adapted via prompt tuning in pipelines such as ThreeStudio.

These models highlight the trade-off between convenience and geometric fidelity. While they offer scalable alternatives to CAD assets, GenAI meshes may contain artifacts such as holes, scale inconsistencies, or noisy surfaces. In this work, we investigate how such imperfections impact downstream 6D pose estimation. Specifically, we evaluate how well a 6D-pose system (FoundationPose) handles these generative meshes under both controlled (LINEMOD) and unconstrained (HOTS) scenarios.

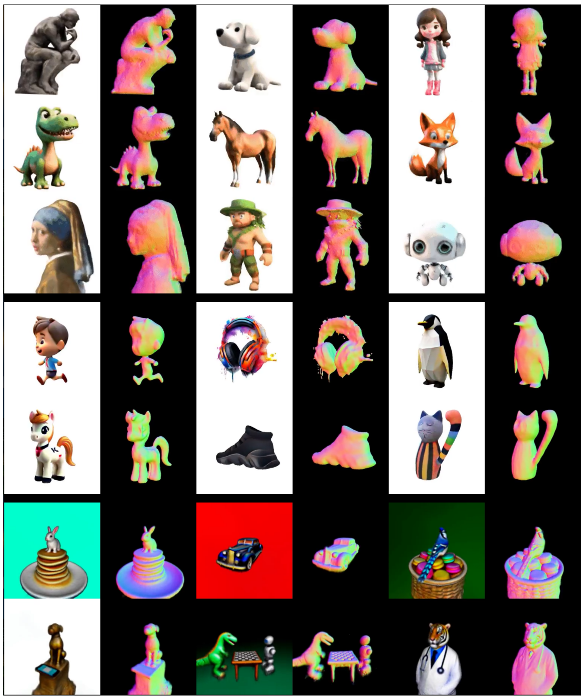


Figure 2.2: Representative single-view 3D reconstructions produced by our three GenAI models (rendered as normal-map overlays). Rows 1–3: Zero123; Rows 4–5: Magic123; Rows 6–7: DreamFusion. Each column is a different object; note the variation in fidelity, scale, and completeness.

2.4 Related Approaches

Several prior works have integrated 3D object models with learning-based pose estimators. PoseCNN (Xiang et al., 2018) uses convolutional networks to regress object pose from RGB input. PVN3D (He et al., 2020) refines keypoints using a hybrid voting strategy on RGB-D data. GDR-Net (Wang et al., 2021) and CosyPose (Labbe et al., 2020) introduce multi-stage optimization frameworks that handle object symmetry and uncertainty. GDR-Net, in particular, demonstrates robustness to occlusion and clutter, which makes it relevant to real-world settings such as those modeled by the HOTS dataset. While generative pipelines have recently demonstrated promising 3D generation quality, few studies evaluate whether these meshes are usable in robotics applications where spatial precision and alignment are critical (Wang et al., 2021), making their downstream applicability to closed-loop tasks an open question.

2.5 Research Gap and Motivation

Despite progress in 3D generation, the downstream utility of GenAI meshes in pose estimation remains underexplored. Specifically, prior research has not systematically evaluated:

- The effect of reconstruction noise or artifacts on 6D pose estimation ((Pöllabauer et al., 2024)).
- The generalization capacity of pose pipelines when working with novel or hallucinated geometries.
- The trade-off between visual realism, mesh structure, and pose accuracy in downstream tasks.

Moreover, many current benchmarks focus on closed-world datasets with clean, well-aligned CAD models. There is a lack of robustness testing across domain shifts, such as from synthetic training conditions to real-world cluttered scenes, a gap highlighted in recent evaluations of generalization limits in pose estimation models (Wang et al., 2021; Xiang et al., 2018). This thesis addresses these gaps by benchmarking the performance of FoundationPose when using GenAI meshes, across both structured datasets like LINEMOD (Hinterstoisser et al., 2012) and real-world, CAD-free settings such as HOTS (Tziafas, 2023).

3 Methods

The experiments in this thesis required both local and high-performance computing environments, along with a diverse software stack involving deep learning, differentiable rendering, and 3D reconstruction toolkits. To keep the main Methods section focused on the conceptual pipeline, all system-level details, including GPU hardware configurations, runtime constraints, environment adaptations, and mesh preprocessing tools, are consolidated in Appendix A.1. Readers interested in reproducibility, deployment feasibility, or infrastructure limitations are encouraged to consult that section.

3.1 Pose Estimation with FoundationPose

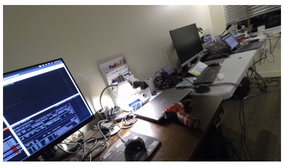
In this study, we use the model-based variant of FoundationPose to estimate 6D object poses from RGB-D input. This mode leverages known 3D meshes to render and refine pose hypotheses, making it directly compatible with CAD models and GenAI-generated meshes. For discussion of the model-free NOF pipeline and the reasons it was excluded from this study, see Appendix A.3.

Rather than treating FoundationPose as a black-box tool, we position it as the central subject of analysis. Our objective is to evaluate its robustness under varying input mesh conditions—comparing clean CAD models, generative AI reconstructions, and controlled noise variants—across datasets with different visual and geometric characteristics. To ensure fair comparison, the internal architecture of FoundationPose remained unchanged, and all experiments used a consistent configuration. Figure A.2 provides a high-level overview of the full track-refine loop used in all trials.

To ensure compatibility with modern GPUs and software stacks, we made several low-level modifications to the FoundationPose environment, including updates to CUDA, PyTorch, and key rendering libraries. These system adaptations are detailed in Appendix A.2.

3.1.1 Required Inputs for Model-based Pose Estimation

FoundationPose's model-based branch needs four core image-/mesh-based inputs (plus camera intrinsics) to generate and refine pose hypotheses. Figure 3.1 shows each at a glance; full descriptions are in Appendix A.4.



(a) RGB image. Appearance comparison (texture, color, contours).

(b) Depth map. Per-pixel distance gives 3D geometry (point clouds, normals).

(c) 3D mesh. Differentiable renderer proxy for synthetic RGB/depth/normal.

(d) Object mask (opt.). Focuses scoring on the object, suppresses background.

Figure 3.1: Key inputs for FoundationPose's model-based pipeline. Camera intrinsics complete the set.

Camera intrinsics. Required to project between pixel and 3D camera rays. Accuracy here is critical for every rendering step.

3.1.2 High-Level FoundationPose Pipeline

FoundationPose transforms an RGB–D frame and a 3D mesh into a final 6D pose through four main stages (Figure 3.2): hypothesis sampling, differentiable rendering, iterative refinement, and hypothesis scoring. For step-by-step details see Appendix A.5.

1. **Pose hypothesis sampling.** Uniformly generate N initial camera–object transformations by sampling viewpoints on an icosphere and applying discrete in-plane rotations.
2. **Differentiable rendering.** For each pose hypothesis, render synthetic RGB, depth, and surface-normal buffers from the 3D mesh via a GPU-accelerated differentiable rasterizer.
3. **Iterative pose refinement.** A Refiner network ingests both rendered and observed buffers to predict incremental rotation ΔR and translation Δt updates for each hypothesis.
4. **Hypothesis scoring and selection.** A Scorer network assigns a confidence score to each refined hypothesis, then selects the highest-scoring pose as the final output.

FoundationPose provides three primary modes of operation for running its 6D pose estimation pipeline. Each mode differs in how it handles input preparation, scene complexity, and intended use cases. While the core architecture, comprising the refiner and scorer networks, remains consistent, the pipelines differ substantially in scale, flexibility, and intended evaluation depth. For additional details on the three modes see Appendix A.6.

3.2 Generative AI Model Selection

Our primary objective was to identify models capable of generating clean, realistic, and geometrically consistent 3D meshes from minimal input, either a single RGB image or a natural language prompt, while maintaining compatibility with the downstream 6D pose estimation framework, FoundationPose.

To achieve this, we adopted the `Threestudio` framework, a unified platform that integrates several state-of-the-art generative pipelines for 3D reconstruction. This framework offered a consistent training interface and robust mesh export capabilities, which allowed us to evaluate different generative architectures under controlled, comparable conditions. Within this environment, we selected three specific methods for in-depth exploration: Zero123, Magic123, and DreamFusion, each adapted via `Threestudio` to support fast training and structured output.

The models were chosen based on the following criteria, each reflecting a technical requirement essential to successful integration with FoundationPose:

- **Mesh Output (OBJ/PLY):** FoundationPose requires explicit 3D geometry as input. Models that produced only point clouds, implicit fields (e.g., NeRFs), or voxel representations were excluded unless they supported reliable mesh extraction.
- **Single-view or Prompt-based Generation:** In keeping with real-world constraints, selected models had to function from minimal input, either a single RGB image or a text prompt, to simulate test-time object capture.
- **Geometric Consistency:** Pose estimation accuracy is highly sensitive to mesh fidelity. Models prone to surface noise, poor topology, or non-manifold geometry were filtered out, as such artifacts degrade downstream scoring and refinement steps.
- **Reasonable Inference Time:** Since our study involved testing multiple object instances, we prioritized models that could complete training and mesh export in a manageable amount of time. Extremely slow pipelines were not feasible given practical constraints.
- **Framework Compatibility:** To streamline integration and reduce tooling friction, all selected models had to run within the same development environment. `Threestudio` fulfilled this requirement by providing consistent interfaces for training, supervision, and mesh export.

Taken together, these criteria ensured that only those generative methods capable of producing high-quality, watertight meshes in a reproducible and efficient manner were considered. The following sections describe how each selected model meets these standards in practice.

3.3 Model Descriptions

For implementation details within the ThreeStudio framework, the underlying optimization loop, and observed trade-offs, please see Appendix A.8, Appendix A.9, and Appendix A.10.

3.3.1 Zero123

Zero123 (also referred to as Zero-1-to-3) is a generative view synthesis model designed to generate novel 2D views of an object from a single RGB image. Originally proposed by Gao et al. (2023), it addresses one of the core challenges in 3D reconstruction: the need for multi-view input to infer consistent geometry. Instead of requiring multiple observations, Zero123 conditions on virtual camera poses, such as azimuth and elevation, and synthesizes how the object would appear from new angles, all without requiring explicit 3D input.

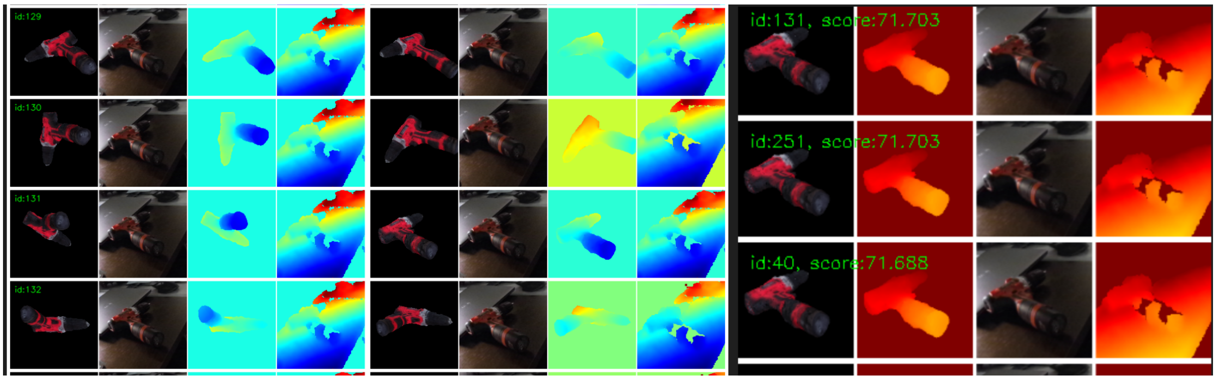


Figure 3.2: Example of FoundationPose on the "drill" scene. Cols 1–4 (initial hypotheses): (1) rendered mesh (textured RGB overlay), (2) observed RGB crop, (3) rendered depth map, (4) colorized (heat-map) depth for visual clarity. Cols 5–8 (after one Refiner step): same four renderings under the updated pose hypothesis. Cols 9–12 (Scorer block): the three highest-scoring hypotheses shown as (9) textured RGB overlay, (10) soft confidence heatmap over the depth, (11) observed RGB, (12) rendered depth. The final output is the hypothesis with the highest overall score.

Unlike earlier works that used Zero123 solely for image generation, this thesis evaluates its use as a standalone model in structured mesh reconstruction. In particular, we test Zero123 on objects from the HOTS dataset to assess whether its pseudo-multiview outputs can support mesh recovery under single-view constraints.

3.3.2 Magic123

Magic123 is a generative framework for 3D reconstruction from a single RGB image and an accompanying text prompt, first introduced by Qian et al. (2023). Unlike Zero123, which only synthesizes novel 2D views, Magic123 directly produces fully textured, watertight 3D meshes. This makes it immediately compatible with downstream applications such as rendering, simulation, and 6D pose estimation.

Its architecture combines image-based and text-conditioned guidance with 3D-aware optimization, allowing it to synthesize explicit 3D geometry from minimal input. This dual conditioning setup, visual and textual, enables Magic123 to produce semantically rich and geometrically consistent reconstructions from scenes where no CAD model is available.

3.3.3 DreamFusion

DreamFusion is a text-to-3D generation framework introduced by Poole et al. (2022), which synthesizes 3D objects directly from natural language prompts using no 3D supervision, multi-view images, or external geometry. It marked a key breakthrough in generative modeling by introducing a novel idea: using a pretrained 2D diffusion model to supervise a 3D representation based on how realistic its renders appear.

At the core of DreamFusion is the Score Distillation Sampling (SDS) technique, where a volumetric NeRF-style scene is optimized based solely on gradients computed by a 2D diffusion model (originally Google's Imagen). Over thousands of iterations, the rendered views from the evolving 3D object are passed through the 2D model, and visual feedback is distilled into geometry, entirely from text.

3.3.4 Mesh Export Pipeline

To ensure consistency across all GenAI meshes, we use ThreeStudio's built-in exporter to produce watertight OBJ/MTL meshes with textures. All models (Zero123, Magic123, DreamFusion) are exported with the same isosurface thresholds, resolution, and texture settings before being fed into FoundationPose. For a full description of the export flags, resolution/threshold sweeps, and the custom batch script we employ, see Appendix A.7.

4 Experimental Setup

4.1 Datasets

In this thesis, FoundationPose is not merely used as a baseline pipeline for 6D pose estimation, it constitutes the primary subject of investigation. Our objective is to rigorously evaluate its reliability, adaptability, and generalization capabilities under varying input conditions, particularly when operating on reconstructed 3D meshes generated by foundation models. To enable this analysis, we adopt a two-pronged dataset strategy, utilizing both a controlled benchmark (LINEMOD) and a challenging, CAD-free scenario (HOTS).

LINEMOD. LINEMOD (Hinterstoisser et al., 2012) is a canonical 6D-pose benchmark comprising

15 distinct objects captured under consistent lighting and background conditions. For each frame it supplies an RGB image, a registered depth map, and a binary instance mask. Camera intrinsics are provided, and every object is paired with a textured 3D model in `.ply` format. These models include precomputed metadata such as object diameter and axis-aligned bounding-box extents. Because LINEMOD delivers this complete set of modalities, images, masks, depth, intrinsics, poses, and high-quality CAD meshes, it serves as an ideal controlled environment for quantifying how substituting AI generated models for ground-truth geometry affects pose-estimation accuracy.

HOTS. The HOTS dataset (Tziafas, 2023) consists of RGB sequences recorded by a robot-mounted camera navigating cluttered indoor scenes, together with per-frame semantic and instance segmentation masks. While original RGB data are provided out of the box, we requested and received monocular depth maps and camera calibration parameters from the HOTS authors to enable 6D-pose inference. Critically, HOTS does not include any canonical 3D object models, making it a natural testbed for evaluating the robustness of pose-estimation pipelines when using AI-generated meshes instead of CAD geometry. This contrast with LINEMOD's controlled completeness allows us to assess generalization under realistic, less curated conditions.

4.2 RQ1 Setup: Adapting HOTS for FoundationPose

Figure A.4 illustrates the end-to-end HOTS restructuring and mesh-preparation pipeline we built on top of FoundationPose's inputs.

To investigate the generalization capabilities of FoundationPose to unstructured, real-world data, we evaluate its performance on the HOTS dataset, which differs significantly from the benchmark LINEMOD dataset. HOTS provides only RGB and masks and lacks depth, intrinsics, and 3D meshes. In RQ1 we address each missing modality in turn:

Depth & Intrinsics. We requested monocular depth maps and camera parameters from the HOTS authors and integrated them without modification.

3D Geometry. We added a generative-AI reconstruction stage to produce textured meshes from single views or prompts.

Data Layout. With all inputs in hand, we reformatted HOTS into the LINEMOD-style directory hierarchy (benchmark vs. demo mode) so FoundationPose can load everything seamlessly.

Inference. Finally, we ran FoundationPose in batch to generate per-frame 4×4 pose matrices, logging them as YAML for downstream evaluation.

Stage 1: Dataset Restructuring and Mesh Preparation. We first convert HOTS into the exact input layout expected by FoundationPose, with two possible modes. (1) A *benchmark-style mode* which reproduces the per-object folder hierarchy of LINEMOD, complete with subfolders for each object's RGB, depth, and mask images, plus metadata files containing camera intrinsics, depth-scale factors, and initial (dummy) poses. (2) a *demo-style mode* which flattens everything into a single folder per object for rapid ad hoc testing, without per-object subdirectories.

This restructuring consists of the following steps:

Directory scaffolding. Automatically create or clear the required folder structure for the selected mode.

Modality standardization. RGB images are copied directly into the output folders; depth arrays in '.npy' are converted to PNG in millimeters; for each object, instance-ID masks are thresholded and saved as binary PNGs at the original resolution.

Metadata insertion. For each frame, write out the camera intrinsics and depth-scale factor; in benchmark-style mode also emit per-object ground-truth pose placeholders.

Mesh normalization. For each 3D model (user-provided or AI-generated), perform the following:

1. **Center at the origin:** Compute the centroid $\mathbf{c} = \frac{1}{N}\sum_{i=1}^{N} \mathbf{v}_i$ then translate each vertex so that the object's center lies at the coordinate $(0,0,0)$: $\mathbf{v}_i^{(c)} = \mathbf{v}_i - \mathbf{c}$.

2. **Optional axis alignment:** If an axis-alignment rotation is required by the user, form $R = R_z(\theta_z)\,R_y(\theta_y)\,R_x(\theta_x)$, then apply $\mathbf{v}_i^{(r)} = R\,\mathbf{v}_i^{(c)}$. Otherwise, set $\mathbf{v}_i^{(r)} = \mathbf{v}_i^{(c)}$.

3. **Uniform scaling:** We first determine how large the mesh is in each direction by finding the minimum and maximum coordinates along the x, y, and z axes. The difference $\max_i v_{i,d}^{(r)} - \min_i v_{i,d}^{(r)}$ gives the size of the mesh along axis d, and taking the maximum of these three values, $\Delta = \max(\Delta_x, \Delta_y, \Delta_z), \quad \Delta_d = \max_i v_{i,d}^{(r)} - \min_i v_{i,d}^{(r)}$ identifies the mesh's largest dimension. If the user has specified a desired real-world length L for that largest side (for example, 0.08 m for an object roughly 8 cm across), we compute a uniform scale factor $s = \frac{L}{\Delta}$. Multiplying every vertex by s, $\mathbf{v}_i^{(s)} = s\,\mathbf{v}_i^{(r)}$, shrinks or enlarges the entire mesh so that its longest side becomes exactly L. In practice, this guarantees all objects, regardless of their

original size, occupy a consistent, user-defined scale in meters.

4. **Export:** Convert units if needed (e.g. meters→millimeters) and write out $\{\mathbf{v}_i^{(s)}\}$ in the required format.

5. **Metadata extraction:** Compute the axis-aligned bounding box $\mathcal{B} = \left[\min_i \mathbf{v}_i^{(s)}, \ \max_i \mathbf{v}_i^{(s)}\right]$ and diameter $\max_{i,j} \|\mathbf{v}_i^{(s)} - \mathbf{v}_j^{(s)}\|$, then record them for evaluation.

Note: Steps 4 and 5 are only performed when generating the LINEMOD-style output; in demo mode, we skip these steps and leave the mesh in its original units and without writing per-model metadata.

Stage 2: Pose Estimation. With HOTS now fully formatted, we run two variants of FoundationPose:

- **Full-dataset mode** processes every object and frame in batch, loading each object's reconstructed (or original) mesh along with its RGB, depth, and mask, and then computing fresh pose predictions for evaluation.
- **Interactive demo mode** steps through a single object's frames in sequence, visualizing the estimated 3D bounding box and local XYZ axes overlaid on the RGB image to inspect orientation quality.

3D Model Generation. To bridge the missing modality, we integrated a generative 3D reconstruction via our ThreeStudio backend. When a new object class is encountered, a representative RGB image or textual prompt is passed to the 3D generator, which outputs a textured mesh. Each generated model is then normalized (centered, optionally rotated, scaled) according to predefined size heuristics. Generated meshes are cached so that an existing file is reused rather than regenerated. External meshes can also be preloaded into a central directory to bypass generation entirely.

Both modes share the same inference core: first, an initial set of pose hypotheses is generated; next, each hypothesis is passed through the refiner network for iterative pose correction; finally, the scorer network assigns a quality score to each refined hypothesis and the best-scoring pose is selected. All inputs and outputs (meshes, images, masks, intrinsics, and pose matrices) are cached so that re-running on unchanged data is effectively instantaneous.

This two-stage pipeline transforms HOTS from an RGB-only collection into a fully compliant evaluation framework for FoundationPose, enabling quantitative assessment of its accuracy and robustness under real-world conditions and with AI-synthesized 3D geometry.

4.3 Preprocessing and Alignment Across All RQs

Because generative 3D models often arrive in arbitrary scale, orientation, and position, we apply a unified, automated normalization and alignment pipeline before any metric computation. All stages below are applied identically in both RQ2 and RQ3, with one minor difference in how we handle centroids:

- **Centroid alignment.**
  - **RQ2:** We translate *both* ground-truth (GT) and AI meshes so that their centers of mass lie at the origin $(0, 0, 0)$.
  - **RQ3:** We leave the GT mesh fixed, and translate only the AI mesh so that its centroid matches the GT centroid.
- **Safe scaling.** To bring both meshes into the same size range, we compute a scale factor
$$s = \left(\frac{V_{\mathrm{GT}}}{V_{\mathrm{AI}}}\right)^{1/3}$$
where V is the enclosed volume of a watertight mesh. If the mesh is not watertight or volume is zero, we fall back to
$$s = \left(\frac{V_{\mathrm{hull,GT}}}{V_{\mathrm{hull,AI}}}\right)^{1/3} \quad \text{or} \quad s = \frac{\|\mathrm{diag}_{\mathrm{GT}}\|}{\|\mathrm{diag}_{\mathrm{AI}}\|},$$
where V_{hull} is the convex-hull volume and $\|\mathrm{diag}\|$ the axis–aligned bounding-box diagonal.
- **Global RANSAC registration.** We first extract matching local features using Fast Point Feature Histograms (FPFH) (Rusu et al., 2009) on two dense point clouds, then run a RANSAC (Fischler and Bolles, 1981) loop to robustly estimate an initial rigid transform $\mathbf{T}$. At each iteration RANSAC:
  1. randomly samples four FPFH-matched point pairs,
  2. solves for the best rotation + translation aligning those pairs,
  3. counts inliers whose transformed source points lie within a threshold $d_{\max}$,
  4. retains the transform with the largest inlier set.

  We only accept $\mathbf{T}$ if its overall fitness exceeds 0.1.

- **Multi-stage ICP refinement.** (Rusinkiewicz and Levoy, 2001) Starting from the RANSAC result, we apply three successive point-to-plane ICP runs, with maximum correspondence distances

$$d_1 > d_2 > d_3$$

to gradually tighten alignment from coarse to sub-millimeter precision.

- **Principal-axis check.** As a diagnostic, we compute each mesh's principal component axes (via PCA on its vertices) and measure the angles between corresponding axes before and after alignment. These angular deviations are not used to drive registration, but confirm that the final pose is both geometrically and rotationally consistent.

Because RQ2 and RQ3 now share this identical pipeline (aside from the centroid step), we will omit redundant detail in the individual sections that follow.

4.4 RQ2 Setup: 3D Reconstruction Accuracy

Figure A.5 shows our RQ2 workflow: pairing each AI-generated mesh with its CAD counterpart, preprocessing, and computing six fidelity metrics.

To quantify how faithfully 3D generative models recover object geometry, we evaluate their output meshes against known CAD ground truth under identical conditions.

Dataset Nine objects spanning industrial and consumer domains are used:

- *LINEMOD* (4): *camera*, *cat*, *duck*, *gorilla.*
- Online (3): *banana*, *mouse*, *soda can.*

Generative Methods & Time Budgets We benchmark the three aforementioned threestudio models: Zero123, Magic123, DreamFusion-IF. Each is allotted three mesh-synthesis budgets: 10 min, 30 min, and 60 min.

Preprocessing & Alignment All meshes undergo the *Preprocessing and Alignment* pipeline (centroid normalization → safe scaling → RANSAC → multistage ICP) described in section 4.1.3.

Metrics On each aligned AI–GT pair we sample dense point clouds $P_{\mathrm{GT}}, P_{\mathrm{AI}}$ and compute six complementary metrics, each probing a different facet of geometric fidelity:

- **Chamfer Distance (CD).** Fan et al. (2017) Captures the average squared distance between the two surfaces, penalizing overall misalignment and missing detail. Formally:

$$\mathrm{CD}(P,Q) = \frac{1}{|P|}\sum_{p\in P}\min_{q\in Q}\|p-q\|^2 + \frac{1}{|Q|}\sum_{q\in Q}\min_{p\in P}\|q-p\|^2.$$

A low CD indicates that most points on one mesh lie close to the other.

- **Earth Mover's Distance (EMD).** (Fan et al., 2017) Measures the minimal "transport cost" to turn one point distribution into the other, thereby capturing global shape differences and mass displacement:

$$\mathrm{EMD}(P,Q) = \min_F \sum_{i,j} F_{ij}\,\|p_i - q_j\|, \quad \text{s.t.} \sum_j F_{ij} = \tfrac{1}{|P|}, \quad \sum_i F_{ij} = \tfrac{1}{|Q|}.$$

EMD is sensitive to large-scale distribution shifts even when CD is small.

- **Hausdorff Distance (HD).** (Hinterstoisser et al., 2012)Reports the worst-case point-to-point deviation, highlighting extreme outliers:

$$\mathrm{HD}(P,Q) = \max\Big\{\max_{p\in P}\min_{q\in Q}\|p-q\|, \max_{q\in Q}\min_{p\in P}\|q-p\|\Big\}.$$

A low HD guarantees no point lies far from its counterpart.

- **Normal Consistency (NC).** (Meyer et al., 2003) Quantifies surface orientation agreement by averaging the cosine similarity of matched normals:

$$\mathrm{NC}(P,Q) = \frac{1}{|P|}\sum_{p\in P}\big|n_{\mathrm{GT}}(p)\cdot n_{\mathrm{AI}}(q^*)\big|, \quad q^* = \arg\min_q \|p-q\|.$$

NC close to 1 indicates almost identical local surface orientation.

- **Mean Curvature Error (MCE).** (Taubin, 1995)Assesses fine-scale surface bending differences: at each sample p, we fit a local PCA to its neighborhood to estimate principal curvatures κ, then average the absolute difference,

$$\mathrm{MCE} = \frac{1}{|P|}\sum_{p\in P}\big|\kappa_{\mathrm{GT}}(p) - \kappa_{\mathrm{AI}}(p)\big|.$$

Small MCE means the AI mesh reproduces local smoothness well.

- **Voxel Intersection-over-Union (IoU).** (Dai et al., 2017) Converts each mesh into a binary voxel grid and computes

$$\mathrm{IoU} = \frac{|\mathrm{vox}_{\mathrm{GT}} \cap \mathrm{vox}_{\mathrm{AI}}|}{|\mathrm{vox}_{\mathrm{GT}} \cup \mathrm{vox}_{\mathrm{AI}}|}.$$

High IoU indicates strong volumetric agreement.

Threshold Selection All rating cutoffs ("excellent"/"good"/"warning"/"bad"/"critical") are defined in Appendix A.12. We aligned them with established norms in 3D reconstruction, and although we computed the full five-level scale for completeness, in the figures below we only annotate the "good" and "bad" thresholds for clarity. Specifically, thresholds were chosen to match prior work on each metric: volumetric IoU (Choy et al., 2016; Wu et al., 2016), Chamfer Distance (Fan et al., 2017), Hausdorff Distance (Rusinkiewicz and Levoy, 2001; Gelfand et al., 2005), Earth Mover's Distance (Fan et al., 2017), Normal Consistency (Tulsiani et al., 2018), and Mean Curvature Error (Pauly et al., 2006).

Visualization In addition to final overlays (GT in green, AI in red), we automatically generate *debug plots* at each pipeline stage such as scale checks, alignment snapshots, and metric error maps. These serve both qualitative inspection and sanity-checking of the normalization and registration steps.

This multi-metric evaluation under identical preprocessing and compute constraints directly addresses RQ2 by revealing each method's geometric fidelity across shape categories and time budgets.

amsmath,amssymb

4.5 RQ3 setup: Pose Estimation Robustness to Mesh Perturbations

Figure A.6 gives the overview of RQ3's two-stage pipeline: AI-mesh vs. noisy-GT evaluation under FoundationPose.

To quantify how 6D pose estimation degrades when the underlying 3D model is imperfect, we evaluate FoundationPose under two perturbation regimes: (1) AI–generated reconstructions and (2) synthetically corrupted ground-truth meshes. This two-pronged approach isolates the effects of coarse geometry errors versus fine-scale noise on downstream pose accuracy.

Input Mesh Variants. We assemble three families of meshes for each object in the LINEMOD benchmark:

- **Ground-Truth (GT):** the LINEMOD CAD models
- **AI-Generated:** from RQ2, the one-hour reconstructions produced by Zero123, Magic123, and DreamFusion-IF, selected for their highest geometric fidelity.
- **Noisy GT:** four controlled perturbations applied to the GT mesh.

Synthetic Noise Injection. Let the original mesh vertices be $\{v_i \in \mathbb{R}^3\}_{i=1}^N$ with outward normals $\{n_i\}$. We produce four corrupted versions $\{v_i'\}$ to mimic different real-world defects:

- **Gaussian Noise (GN).** (Qi et al., 2017a) We add small, independent zero-mean displacements to every vertex, simulating uniform measurement jitter:

  $$v_i' \;=\; v_i + \varepsilon_i, \quad \varepsilon_i \sim \mathcal{N}\big(0,\; \sigma^2 I_3\big), \quad \sigma = 0.1.$$

  Here each ε_i is drawn independently for the x, y, z coordinates.

- **Normal-directional Noise (NN).** To imitate depth-sensor noise that primarily perturbs along surface normals, we displace each vertex by a scalar along its normal:

  $$v_i' \;=\; v_i + \varepsilon_i\, n_i, \quad \varepsilon_i \sim \mathcal{N}(0,\; \sigma^2), \quad \sigma = 0.1.$$

  Each ε_i is a single scalar offset sampled independently for each vertex.

- **Speckle Noise (SN).** We multiply each coordinate by a small random factor, modeling multiplicative gain errors that grow with distance from the origin:

  $$v_i' \;=\; v_i + v_i \odot \varepsilon_i, \quad \varepsilon_i \sim \mathcal{N}\big(0,\; \sigma^2 I_3\big), \quad \sigma = 0.1.$$

  Thus each component of v_i is perturbed proportionally to its own magnitude.

- **Outlier Noise (ON).** (Qi et al., 2017b) To simulate sparse, large-magnitude defects (e.g. bad scan points), we pick 2% of vertices at random and apply a larger Gaussian jump:

  $$v_i' = \begin{cases} v_i + \delta_i, & \delta_i \sim \mathcal{N}(0,\; \sigma^2 I_3), \quad i \in \mathcal{O}, \\ v_i, & \text{otherwise}, \end{cases}$$

  where $\mathcal{O} \subset \{1, \ldots, N\}$ with $|\mathcal{O}|/N = 0.02$ and $\sigma = 0.1$.

Pose Estimation Pipeline. For every mesh variant (GT, AI, and each noise type), we:

1. **Normalize & Align:** apply the shared preprocessing to match the dataset's canonical frame (Section 4.3).

2. **Dataset Restructuring:** regenerate the LINEMOD-style folder hierarchy so that each mesh seamlessly replaces the CAD model in inference.

3. **Inference:** run FoundationPose in batch mode, loading RGB, depth, mask, intrinsics, and the test mesh to produce per-frame 4×4 pose matrices T_{pred}.

4. **Logging:** emit YAML files containing the full transformation matrix (pose) for each frame, to be consumed by the evaluation pipeline.

Evaluation and Metrics. We post-process all predicted versus ground-truth poses using four complementary error measures:

- **Rotation Error.** (Mur-Artal et al., 2015) Measures the smallest angle (in degrees) required to rotate the predicted orientation to the ground-truth orientation. If R_{GT} and R_{pred} are the 3×3 rotation matrices, then

$$E_R = \arccos\Bigl(\frac{\operatorname{tr}\bigl(R_{\text{GT}}^{\top} R_{\text{pred}}\bigr) - 1}{2}\Bigr) \quad [\text{deg}],$$

  where $\operatorname{tr}(\cdot)$ denotes the matrix trace (sum of diagonal entries).

- **Translation Error.** (Mur-Artal et al., 2015) Computes the Euclidean distance between the predicted and ground-truth position vectors. If t_{GT}, $t_{\text{pred}} \in \mathbb{R}^3$, then

$$E_t = \| t_{\text{GT}} - t_{\text{pred}} \|_2 \quad [\text{m}],$$

  where $\| \cdot \|_2$ is the standard 2-norm.

- **Pose Error.** Aggregates rotation and translation discrepancies into a single scalar by taking the Frobenius norm (Horn and Johnson, 2012) of the difference between the two 4×4 homogeneous transform matrices:

$$E_P = \bigl\| T_{\text{GT}} - T_{\text{pred}} \bigr\|_F,$$

  where Frobenius norm is defined as

$$\|A\|_F = \sqrt{\sum_{i,j} A_{ij}^2} = \sqrt{\operatorname{tr}\bigl(A^{\top} A\bigr)}.$$

- **Average Distance of Model Points (ADD) Error.** (Hinterstoisser et al., 2012) Computes the average Euclidean distance between each model point transformed by the predicted versus ground-truth pose. If $\{x_i\}_{i=1}^{M}$ are the 3D model points, then

$$E_{\text{ADD}} = \frac{1}{M} \sum_{i=1}^{M} \bigl\| R_{\text{pred}}\, x_i + t_{\text{pred}} - (R_{\text{GT}}\, x_i + t_{\text{GT}}) \bigr\|_2.$$

All errors are aggregated into per-object, per-mesh-type CSV tables and accompanied by debug plots (error histograms, trend lines) and 3D alignment snapshots. This rigorous setup directly addresses RQ3 by revealing how pose accuracy deteriorates under varied mesh quality.

5 Results and Interpretation

5.1 RQ1: Generalization and Reliability

Figure 5.1 shows one full inference pass on a HOTS frame (the Pringles object), organized into three logical blocks of columns:

Refiner hypotheses (columns 1–4 and 5–8): We display two groups of four renderings per pose hypothesis, "before" refinement (cols 1–4) and "after" one refinement step (cols 5–8):

- *Cols 1 & 5:* Rendered mesh overlay.
- *Cols 2 & 6:* Observed RGB crop.
- *Cols 3 & 7:* Synthetic depth map.
- *Cols 4 & 8:* Colorized (heat-map) depth.

Notice how the "after" columns (5–8) bring the mesh silhouette and depth much closer to the real observation.

Scorer block (columns 9–12)

- *Cols 9 & 10:* Top-ranked hypotheses, shown as (i) RGB overlay and (ii) confidence heatmap over the depth.
- *Cols 11 & 12:* Ground-truth RGB and its registered depth for direct comparison.

Finally, the far right (not shown in this grid) is the single pose selected by the Scorer, plotted as a green 3D bounding box with red/green/blue XYZ axes on the original RGB image.

This layout illustrates both the Refiner's ability to tighten coarse hypotheses against the true pose and the Scorer's capacity to identify the optimal alignment using real RGB–D observations. Crucially, it confirms that FoundationPose can operate on novel, unstructured scenes without any ground-truth CAD geometry. These qualitative findings reveal several key behaviors:

Strong coarse-to-fine alignment. Even though the Refiner's initial hypotheses (cols 1–4) are sampled uniformly around the object and often bear little resemblance to the true pose, a single refinement iteration (cols 5–8) consistently "snaps" these hypotheses into tight overlap with both the RGB silhouette and the observed depth. For example, in the "keyboard" runs, you can see how the flat rectangular mesh originally appears tilted and

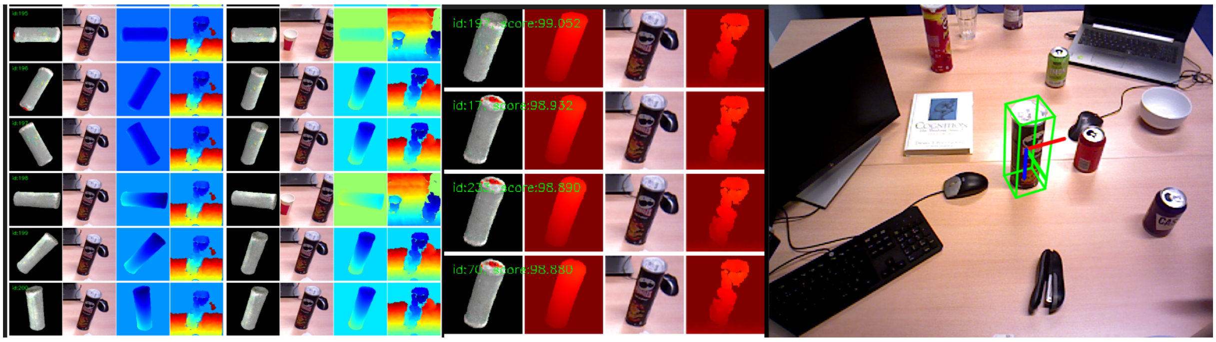


Figure 5.1: Three stages of FoundationPose on a HOTS frame for the *Pringles* object: (left) initial vs. refined hypotheses, (center) top-scoring hypotheses, (right) final selected pose. For further qualitative examples, please see Appendix A.13.

mis-scaled, but post-refinement it matches both the RGB and depth outlines almost perfectly. This behavior confirms that the learned refiner network is robust to large initial errors and capable of leveraging both modalities to recover a plausible alignment.

Reliable hypothesis ranking. The Scorer block (cols 9–12) then selects the best-aligned hypotheses by comparing the refined renderings against the true RGB and depth observations. In our examples, the top-ranked pose almost always corresponds to the visually best match and discards nearby spurious alignments. This demonstrates that the scorer's confidence heatmaps are well calibrated, even in cluttered scenes with heavy occlusion, allowing the system to pick the correct orientation from among multiple plausible candidates.

Accurate final overlays. When plotted as a 3D bounding box with RGB-coded axes (red = X, green = Y, blue = Z) on the original image, the chosen poses show only millimeter-level misalignment in translation and a few degrees of angular error in rotation. In the "scissors" example (Appendix A.13), despite the object's thin geometry and glossy surface, the final overlay remains tightly bound to the real silhouette, confirming that FoundationPose can handle challenging thin-walled objects without CAD models.

Failure modes & sensitivity. We do observe occasional residual jitter on highly symmetric or textureless surfaces (e.g. the cylindrical Pringles can), where multiple poses can explain the same depth silhouette. In those cases, the scorer sometimes hesitates between two equally plausible orientations, which manifests as slight flipping of the long axis. Still, the error remains within a few degrees and millimeters, acceptable for many downstream applications.

Collectively, these observations demonstrate that, even when restricted to RGB, depth, and mask inputs and fed AI-generated meshes, FoundationPose reliably recovers accurate 6D poses on previously unseen objects with no manual intervention.

5.2 RQ2: Accuracy and Trade-offs in GenAI 3D Models

Figures 5.2–5.7 summarize six geometric-fidelity metrics for DreamFusion, Magic123, and Zero123 across all objects [‡]

- **Chamfer Distance** (lower is better)
- **Earth Mover's Distance** (lower is better)
- **Hausdorff Distance** (lower is better)
- **Mean Curvature Error** (lower is better)
- **Normal Consistency** (higher is better)
- **Voxel IoU** (higher is better)

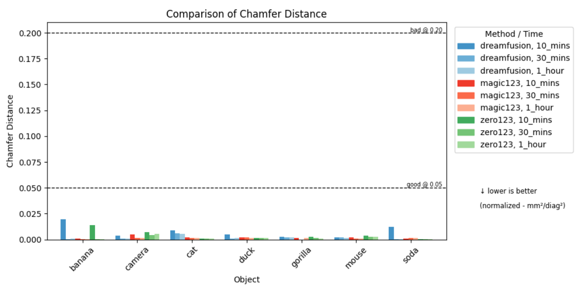


Figure 5.2: Chamfer Distance for all reconstructions and time budgets (normalized $\mathrm{mm}^2/\mathrm{diag}^2$). Lower is better.

[‡]Full prompts and input images in Appendix A.14; reconstruction examples in Appendix A.15; per-object plots in Appendix A.16.

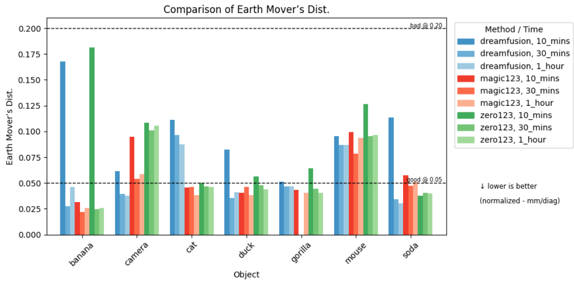


Figure 5.3: Earth Mover's Distance for all reconstructions and time budgets (normalized mm/diag**). Lower is better.**

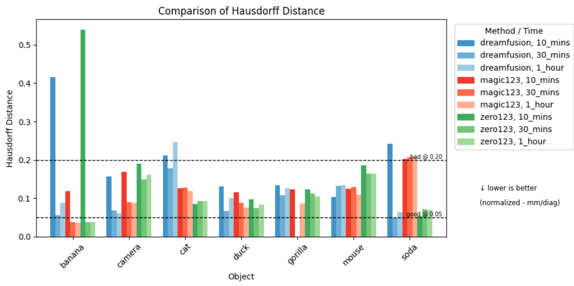


Figure 5.4: Hausdorff Distance for all reconstructions and time budgets (normalized mm/diag**). Lower is better.**

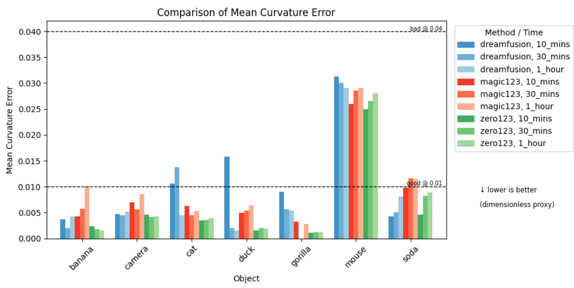


Figure 5.5: Mean Curvature Error for all reconstructions and time budgets (dimensionless proxy). Lower is better.

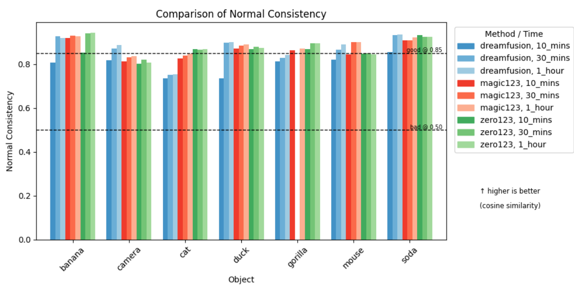


Figure 5.6: Normal Consistency for all reconstructions and time budgets (cosine similarity). Higher is better.

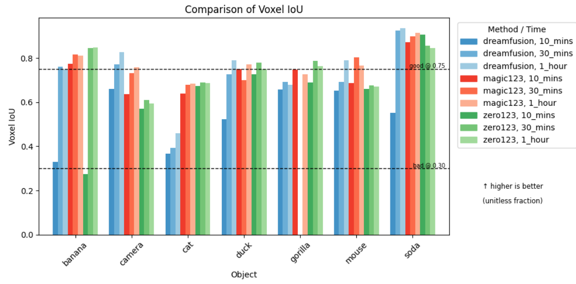


Figure 5.7: Voxel Intersection-over-Union for all reconstructions and time budgets (unitless fraction). Higher is better.

As these plots demonstrate, no single pipeline dominates across every metric; instead, each method exhibits a distinct compute-quality trade-off that also depends on object complexity. (see Appendix A.17 for a per-object "vote" analysis and further discussion).

First, both Chamfer and Earth Mover's distances (Figures 5.2 and 5.3) show that DreamFusion and Zero123 begin with relatively high error at 10 min, reflecting noisy initial volumes, but decrease sharply by 30 min and again by 60 min. Magic123, by contrast, achieves near-"good" Chamfer and EMD thresholds even at 10 min, with only marginal gains thereafter.

The more outlier-sensitive Hausdorff distance (Figure 5.4) underscores that short runs often leave extreme pointwise discrepancies unresolved. By 30 min, most pipelines attain errors between the "good" and "bad" lines, and by 60 min outlier spikes are largely tamed, although Magic123 shows occasional over-smoothing (e.g., its soda-can rim).

Mean Curvature Error (Figure 5.5) further reveals that DreamFusion steadily improves surface smoothness with time, whereas Magic123 and Zero123 may incur slight increases in curvature error at longer runtimes, hinting at subtle surface artifacts from extended sampling.

Normal Consistency and Voxel IoU (Figures 5.6 and 5.7) both rise with synthesis time. By 60 min, simple, symmetric objects (e.g., duck, soda can) reach near-perfect scores, while highly articulated shapes like the camera and cat remain challenging, especially for DreamFusion, which struggles on the Cat prompt even after an hour.

Together, these results indicate that for rapid prototyping, Magic123 offers the best balance of speed and fidelity, delivering "good" reconstructions within 10 min. When maximal accuracy is required and computation time is available, DreamFusion or Zero123 are preferable, reliably achieving CAD-level geometry after 30–60 min. For a full per-object breakdown and "vote" analysis, see Appendix A.16.

5.3 RQ3: Robustness to Mesh Perturbations

Figure 5.8–5.12 show per-object pose errors (rotation, translation, combined pose, ADD) for five LINEMOD objects (IDs 1 = gorilla, 4 = camera, 6 = cat, 9 = duck, 10 = egg carton) under eight mesh variants:

- **Ground-Truths (“original”)**
- **Synthetic noise:** Gaussian, Normal, Speckle, Outlier
- **AI reconstructions:** Zero123, Magic123, DreamFusion

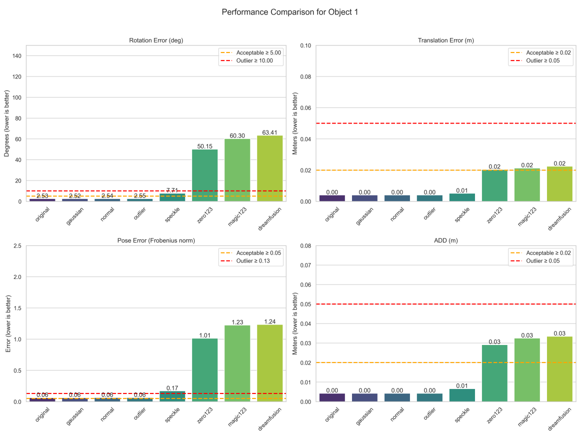


Figure 5.8: Pose-estimation errors for LINEMOD Object 1 (Gorilla) under noisy and AI-generated meshes. Dashed lines mark “acceptable” (orange) and “outlier” (red) thresholds for each metric.

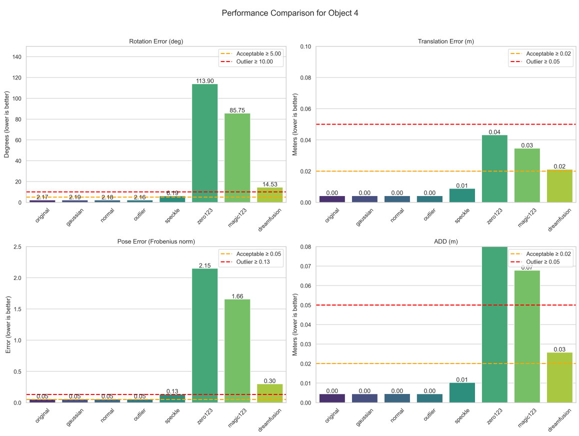


Figure 5.9: Pose-estimation errors for LINEMOD Object 4 (Camera) under noisy and AI-generated meshes.

Figure 5.10: Pose-estimation errors for LINEMOD Object 6 (Cat) under noisy and AI-generated meshes. The DreamFusion result was corrupted during export and is therefore omitted.

Figure 5.11: Pose-estimation errors for LINEMOD Object 9 (Duck) under noisy and AI-generated meshes.

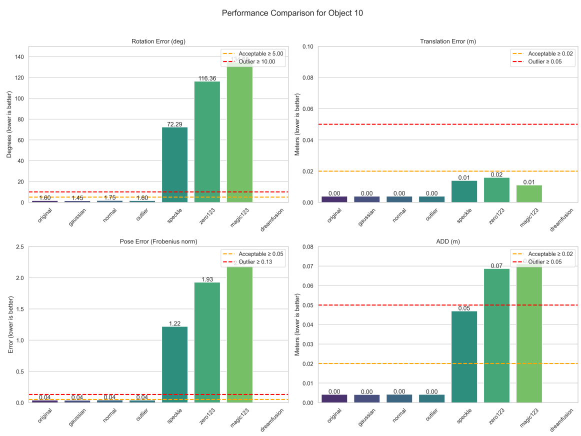


Figure 5.12: Pose-estimation errors for LINEMOD Object 10 (Egg carton) under noisy and AI-generated meshes. The DreamFusion result was corrupted during export and is therefore omitted.

Across mild noise cases (Gaussian, Normal, Outlier), errors remain near zero; Speckle introduces only small deviations ($<7°$ rotation, <10 mm ADD). In contrast, all GenAI meshes yield large pose failures (tens of degrees, centimeters of translation, and multi-cm ADD), with Zero123 $>$ Magic123 $>$ DreamFusion in that order.

Together, these quantitative results reveal four key behaviors of FoundationPose under mesh perturbations:

Robustness to small jitter and sparse spikes. Gaussian and Normal noise, random small offsets in arbitrary directions or along each surface normal, and sparse Outlier noise, occasional large jumps on a few vertices, all have negligible impact on final pose ($\Delta R < 0.1°$, ADD $< 1\,\text{mm}$). This confirms that FoundationPose's refinement stage easily absorbs minor, distributed perturbations and ignores isolated vertex spikes.

Moderate impact of scaling distortions. Speckle noise (multiplicative scaling away from the object center) raises rotation errors to $\sim$5–7° and ADD to $\sim$10 mm, yet remains under our "outlier" thresholds. The uniform stretching slightly challenges refinement but does not break alignment.

Severe failure on coarse reconstructions. All GenAI-generated meshes incur large errors, tens of degrees in rotation, centimeters in translation, and multiple-centimeter ADD, far beyond "acceptable" lines. DreamFusion performs worst, Magic123 is intermediate, and Zero123 is best. Coarse geometric flaws cannot be corrected by refinement alone.

Rotation error dominates. Across both synthetic noise and AI variants, angular misalignment is the main contributor to the combined Frobenius-norm pose error; translation and ADD grow more slowly, underscoring that incorrect orientation is the principal failure mode.

Together, these results demonstrate that FoundationPose tolerates small, localized surface noise but breaks down catastrophically on coarse generative meshes. For detailed per-method outlier trends on the Duck object, see Appendix A.18. However, we also got two striking examples, Figure A.28 and Figure A.29, where DreamFusion's camera mesh and Zero123's cat mesh each achieve near-threshold performance (just under our 10° rotation limit) for long sequences of frames. These "almost successful" stretches suggest that, with the right input image or prompt, GenAI reconstructions could produce meshes suitable for 6D pose estimation.

6 Conclusion

In this thesis, we have demonstrated that FoundationPose can be seamlessly extended to operate on AI-generated meshes (Zero123, Magic123, DreamFusion) by building a unified preprocessing pipeline that ingests single-view RGB and depth inputs or CAD models and emits the exact directory structure and data formats required by NVIDIA's system. Through our zero-shot experiments on the HOTS dataset, we showed that FoundationPose reliably recovers accurate 6D poses in cluttered, CAD-free scenes using only RGB, mask, depth, and reconstructed meshes, confirming its remarkable generalization capacity. A multi-metric fidelity benchmark across six complementary geometry measures (Chamfer, Earth Mover's, Hausdorff, mean curvature, normal consistency, and voxel IoU) and three compute budgets (10, 30, and 60 min) revealed the compute-quality trade-offs of each generative method, guiding practitioners on when to choose rapid-prototyping (Magic123) versus high-fidelity reconstruction (DreamFusion or Zero123). Our robustness analysis further quantified FoundationPose's tolerance to small surface jitters, Gaussian, normal, speckle, and sparse outliers, while highlighting its catastrophic failure on overly coarse or topologically flawed meshes, with angular misalignment emerging as the principal source of error. Combined, these findings unite into a critical insight: FoundationPose's impressive scene-level generalization is conditional on having a sufficiently faithful mesh proxy. In practice, this translates into actionable recommendations, use Magic123 for quick turnarounds, allocate at least 30 min of synthesis for DreamFusion or Zero123 when accuracy is paramount, and always verify that your mesh preserves global shape and scale before pose estimation. Looking forward, we believe that tighter "mesh-pose" co-training, where refinement networks jointly correct both pose and geometry, and the incorporation of geometry-aware priors into generative reconstructions will be key to unlocking truly zero-shot, plug-and-play 6D tracking on arbitrary objects and scenes. Despite these encouraging outcomes, our work is subject to several practical and methodological limitations.

Limitations. For RQ1, our ability to evaluate generative-AI reconstruction on the HOTS dataset was constrained by hardware and software availability. In our Habrok environment, Docker could not be installed, forcing us to run the end-to-end pipeline on a local workstation with only 8 GB of GPU memory. As a result, we were able to integrate and test only DreamFusion-SD (the lower-quality variant of DreamFusion) rather than the full suite of 3D methods. We chose DreamFusion-SD to validate that the pipeline functions as intended, but its lower fidelity may understate the potential of higher-capacity generative models. Encouragingly, successful integration of DreamFusion-SD suggests that, given access to a more powerful GPU, the same framework could readily accommodate other reconstructions. A second limitation in RQ1 concerns background removal: although we implemented a custom segmentation-based pipeline, ThreeStudio's model often reinserted spurious background pixels despite apparent masking. To guarantee clean inputs, we ultimately relied on an external web service, ClipDrop, to strip backgrounds before feeding images into ThreeStudio, introducing an extra manual step and dependence on proprietary tooling.

In RQ2, our reconstruction-accuracy benchmarks were similarly affected by data availability. The LINEMOD dataset does not provide high-resolution (1024×1024) images of its objects, and we lacked physical access to better-lit or higher-fidelity captures. To compensate, we used a state-of-the-art generative-AI image model (GPT-4o) to upsample and synthetically recreate each LINEMOD object at 1024×1024 resolution. While this approach produced visually convincing inputs, it may also bias 3D synthesis methods toward artefacts or textures introduced by the image generator rather than real-world details. We applied the same AI-based image augmentation procedure to out-of-LINEMOD objects (e.g. banana, soda can) to maintain procedural consistency, but this uniformity may mask differences between authentic photographs and AI outputs.

The principal limitation in RQ3 arises from the ambiguity of "canonical orientation" in 3D generative modeling. Unlike ground-truth CAD models, which implicitly define an object's default pose, AI-generated meshes often arrive in arbitrary rotations. For instance, one user might consider a water bottle's natural orientation to be vertical, while another expects it to lie horizontally. Our evaluation defines ground-truth orientation by the centroid-based alignment of the CAD mesh, and we translate AI meshes to match that centroid. However, this convention may not reflect an intrinsic "correct" orientation for all objects, and differing semantic interpretations of an object's upright pose could introduce systematic bias into pose-error metrics.

Together, these limitations highlight clear paths for advancing our work, scaling the pipeline onto more capable hardware, automating robust background subtraction, sourcing or capturing higher-resolution imagery, and developing orientation-invariant evaluation criteria, and point toward concrete next steps for extending and strengthening our 6D pose estimation framework.

Future Research. First, in the context of RQ1, we envision a fully automated input-preparation workflow that requires only a single RGB image. Recent advances demonstrate that instance masks can be reliably extracted with Mask R-CNN (He et al., 2017), and dense depth estimates can be obtained via monocular depth networks such as MiDaS (Ranftl et al., 2022). Combining these with our existing 3D-generation stage (e.g. Zero123, DreamFusion variants) and a single-image camera-calibration algorithm (Criminisi et al., 2000), one could automatically produce the complete set of inputs, mask, depth, mesh, and intrinsics, required by FoundationPose. Such an end-to-end pipeline would greatly simplify data collection, enabling direct inference from everyday photographs or video frames without manual preprocessing.

Second, for RQ2, we identify opportunities to accelerate and diversify 3D generative workflows. One approach is to use the results of a low-budget (e.g. 1 k-iteration) diffusion run as training data for a lightweight neural surrogate that approximates a full (10 k-iteration) model, dramatically reducing inference time and resource consumption while preserving fidelity. Additionally, we recommend systematically evaluating the impact of input-image resolution, since ThreeStudio resizes all inputs to 128×128, it is important to understand how lower-resolution imagery affects reconstruction quality across different iteration budgets. Prompt engineering also warrants deeper study: our preliminary tests showed that Magic123 benefits from broad 2D context, whereas detailed textual prompts improved DreamFusion outputs. Automating the joint optimization of image–prompt pairs (or even combining DreamFusion and Zero123 into a hybrid model) represents a promising avenue. Finally, extending beyond the current methods to newer 3D-generation frameworks (e.g. One-2-3-45 (Liu et al., 2023), VolRecon (Ranftl et al., 2022)) could reveal further gains in geometric accuracy and computational efficiency.

Finally, in RQ3, we highlight two key areas for future exploration. First, more aggressive or structurally informed noise models, beyond the gentle Gaussian, normal, speckle, and sparse-outlier perturbations, could stress-test pose estimators under extreme surface and topology deviations. Second, the challenge of aligning AI-generated meshes to a canonical orientation and centroid remains unresolved: arbitrary rotations in generative outputs can introduce systematic bias in pose metrics. Future work might develop orientation-invariant evaluation criteria or leverage prompt-based control to steer generated meshes toward a desired "upright" pose. Additionally, investigating the necessity of the refiner stage, by comparing inference directly on initial random pose hypotheses versus post-refinement, could clarify the trade-offs between speed and accuracy in 6D pose estimation.

These avenues promise to enhance the robustness, speed, and autonomy of 6D pose estimation pipelines in real-world settings.

References

Besl, P. J. and McKay, N. D. (1992). A method for registration of 3-d shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(2):239–256.

Brachmann, E., Krull, A., Michel, F., Gumhold, S., Shotton, J., and Rother, C. (2014). Learning 6d object pose estimation using 3d object coordinates. In *European Conference on Computer Vision (ECCV)*.

Choy, C. B., Xu, D., Gwak, J., Chen, K., and Savarese, S. (2016). 3d-r2n2: A unified approach for single and multi-view 3d object reconstruction. In *Computer vision–ECCV 2016: 14th European conference, amsterdam, the netherlands, October 11-14, 2016, proceedings, part VIII 14*, pages 628–644. Springer.

Criminisi, A., Reid, I., and Zisserman, A. (2000). Single view metrology. *International Journal of Computer Vision*, 40:123–148.

Curless, B. and Levoy, M. (1996). Volumetric methods for visual hulls. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 303–312.

Dai, A., Ruizhongtai Qi, C., and Nießner, M. (2017). Shape completion using 3d-encoder-predictor cnns and shape synthesis. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 5868–5877.

Fan, H., Su, H., and Guibas, L. J. (2017). A point set generation network for 3d object reconstruction from a single image. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 605–613.

Fischler, M. A. and Bolles, R. C. (1981). Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395.

Gao, R., Lin, T., Zhang, X., Wang, Y., Xie, Y., and Song, J. (2023). Zero-1-to-3: Zero-shot one image to 3d object. *arXiv preprint arXiv:2303.11328*.

Gelfand, N., Mitra, N. J., Guibas, L. J., and Pottmann, H. (2005). Robust global registration. In *Symposium on geometry processing*, volume 2, page 5. Vienna, Austria.

He, K., Gkioxari, G., Dollár, P., and Girshick, R. (2017). Mask r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 2961–2969.

He, Y., Sun, W., Huang, H., Liu, J., Fan, H., and Sun, J. (2020). Pvn3d: A deep point-wise 3d keypoints voting network for 6dof pose estimation. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Hinterstoisser, S., Lepetit, V., Ilic, S., Holzer, S., Bradski, G., Konolige, K., and Navab, N. (2012). Model based training, detection and pose estimation of texture-less 3d objects in heavily cluttered scenes. In *Asian Conference on Computer Vision (ACCV)*.

Hodan, T., Haluza, P., et al. (2017). T-less: An rgb-d dataset for 6d pose estimation of texture-less objects. In *2017 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 880–888. IEEE.

Hodan, T., Matas, J., Obdrzalek, S., Michel, F., et al. (2018). Bop: Benchmark for 6d object pose estimation. In *European Conference on Computer Vision (ECCV)*.

Horn, R. A. and Johnson, C. R. (2012). *Matrix analysis*. Cambridge university press.

Labbe, Y., Carpentier, J., Aubry, M., and Sivic, J. (2020). Cosypose: Consistent multi-view multi-object 6d pose estimation. In *European Conference on Computer Vision (ECCV)*, pages 574–591. Springer.

Lepetit, V., Moreno-Noguer, F., and Fua, P. (2009). Epnp: An accurate o(n) solution to the pnp problem. *International journal of computer vision*, 81:155–166.

Liu, M., Xu, C., Jin, H., Chen, L., Varma T, M., Xu, Z., and Su, H. (2023). One-2-3-45: Any single image to 3d mesh in 45 seconds without per-shape optimization. *Advances in Neural Information Processing Systems*, 36:22226–22246.

Meyer, M., Desbrun, M., Schröder, P., and Barr, A. H. (2003). Discrete differential-geometry operators for triangulated 2-manifolds. In *Visualization and mathematics III*, pages 35–57. Springer.

Mur-Artal, R., Montiel, J. M. M., and Tardos, J. D. (2015). Orb-slam: A versatile and accurate monocular slam system. *IEEE transactions on robotics*, 31(5):1147–1163.

Pauly, M., Kobbelt, L. P., and Gross, M. (2006). Point-based multiscale surface representation. *ACM Transactions on Graphics (TOG)*, 25(2):177–193.

Pöllabauer, T., Brunhuber, J., Tusch, B., and Seitz, S. (2024). Improving 6d object pose estimation of metallic household and industry objects. *arXiv preprint arXiv:2503.03655*.

Poole, B., Jain, A., Barron, J. T., and Mildenhall, B. (2022). Dreamfusion: Text-to-3d using 2d diffusion. *arXiv preprint arXiv:2209.14988*.

Qi, C. R., Su, H., Mo, K., and Guibas, L. J. (2017a). Pointnet: Deep learning on point sets for 3d classification and segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 652–660.

Qi, C. R., Yi, L., Su, H., and Guibas, L. J. (2017b). Pointnet++: Deep hierarchical feature learning on point sets in a metric space. *Advances in neural information processing systems*, 30.

Qian, G., Zhang, C., Xu, B., Xu, Y., Ma, K.-Y., and Wang, W. (2023). Magic123: One image to high-quality 3d object generation with diffusion prior. *arXiv preprint arXiv:2306.17843*.

Ranftl, R., Lasinger, K., Hafner, D., Schindler, K., and Koltun, V. (2022). Towards robust monocular depth estimation: Mixing datasets for zero-shot cross-dataset transfer. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(3).

Rusinkiewicz, S. and Levoy, M. (2001). Efficient variants of the icp algorithm. In *Proceedings third international conference on 3-D digital imaging and modeling*, pages 145–152. IEEE.

Rusu, R. B., Blodow, N., and Beetz, M. (2009). Fast point feature histograms (fpfh) for 3d registration. In *2009 IEEE international conference on robotics and automation*, pages 3212–3217. IEEE.

Schönberger, J. L. and Frahm, J.-M. (2016). Structure-from-motion revisited. In *CVPR*.

Seitz, S. M., Curless, B., et al. (2006). A comparison and evaluation of multi-view stereo reconstruction algorithms. In *CVPR*.

Tatarchenko, M., Dosovitskiy, A., and Brox, T. (2016). Multi-view 3d models from single images with a convolutional network. *European Conference on Computer Vision (ECCV)*.

Taubin, G. (1995). Estimating the tensor of curvature of a surface from a polyhedral approximation. In *Proceedings of IEEE International Conference on Computer Vision*, pages 902–907. IEEE.

Threestudio (2023). Threestudio: A unified framework for single-image and text-to-3d generation. `https://github.com/threestudio-project/threestudio`. Accessed: 2025-05-18.

Tulsiani, S., Efros, A. A., and Malik, J. (2018). Multi-view consistency as supervisory signal for learning shape and pose prediction. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2897–2905.

Tziafas, G. (2023). Hots dataset for rgb-d object pose estimation. `https://github.com/gtziafas/HOTS`. Accessed: 2025-05-12.

Wang, C., Cai, Y., Lin, S., Qian, C., Ouyang, W., Loy, C. C., and Liu, X. (2021). Gdr-net: Geometry-guided direct regression network for monocular 6d object pose estimation. In *CVPR*.

Wen, B., Yang, W., Kautz, J., and Birchfield, S. (2024). Foundationpose: Unified 6d pose estimation and tracking of novel objects. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 17868–17879.

Wu, J., Zhang, C., Xue, T., Freeman, B., and Tenenbaum, J. (2016). Learning a probabilistic latent space of object shapes via 3d generative-adversarial modeling. *Advances in neural information processing systems*, 29.

Xiang, Y., Schmidt, T., Narayanan, V., and Fox, D. (2018). Posecnn: A convolutional neural network for 6d object pose estimation in cluttered scenes. In *Robotics: Science and Systems (RSS)*.

A Appendix

A.1 Hardware & System Adaptations

A.1.1 Hardware and Computational Setup

The experiments conducted in this thesis required both local and high-performance computing resources due to the memory-intensive nature of generative 3D reconstruction.

Initial testing was carried out on a personal laptop equipped with an NVIDIA RTX 4070 Laptop GPU (8 GB VRAM). While suitable for debugging and low-resolution trials, this setup was insufficient for full 3D mesh optimization, especially during texture baking and score distillation sampling (SDS). Out-of-memory (OOM) errors frequently occurred when using larger batch sizes or higher resolutions, making it impossible to run multi-stage pipelines like Magic123 end-to-end.

To address these limitations, the project transitioned to the Hábrók High-Performance Computing (HPC) cluster at the University of Groningen. This provided access to nodes with high-memory GPUs, including:

- NVIDIA A100 (40 GB VRAM)
- NVIDIA V100 (32 GB VRAM)

The increased memory capacity enabled:

- Full training of all three generative models (Zero123, Magic123, and DreamFusion)
- Higher image resolutions and stronger guidance weights during SDS
- Stable mesh generation and texture export without crashing

All model generation experiments, including training and mesh export using Zero123, Magic123, and DreamFusion, were executed on Hábrók. These required large memory capacity and stable runtime environments. In contrast, FoundationPose integration and pose estimation evaluations were performed locally on an NVIDIA RTX 4070 Laptop GPU, as they were comparatively lightweight in terms of computational requirements.

A.1.2 Software Stack and Pipeline Tools

The computational pipeline relied on several key software frameworks and toolkits:

- **FoundationPose:** NVIDIA's model-based 6D object pose estimator, used for evaluation.
- **threestudio:** Unified framework used to train and export 3D meshes from Zero123, Magic123, and DreamFusion variants.
- **PyTorch:** Primary deep learning framework for all training loops.
- **Kaolin & PyTorch3D:** Used for differentiable mesh rendering, voxelization, and geometry preprocessing.
- **nvdiffrast:** GPU rasterizer used within FoundationPose for synthetic view generation.
- **Docker & Conda:** Used for environment isolation across heterogeneous systems.

Integration was handled modularly, with the output of each generative method exported as an OBJ+MTL+texture triplet and passed to FoundationPose via a standardized preprocessing interface.

A.1.3 Memory and Runtime Constraints

Each generative method introduced unique memory and runtime considerations, summarized below:

- **Zero123 (threestudio):** ∼12–32 GB VRAM, ∼15-20 minutes per object
- **Magic123 (coarse only):** ∼12 GB VRAM, ∼30–40 minutes per object
- **DreamFusion (DeepFloyd IF):** ∼20 GB VRAM, ∼30–40 minutes

We omitted the official full Magic123 pipeline due to instability and excessive memory demands. Similarly, DreamFusion runs were limited to coarse optimization stages, avoiding full 3D diffusion-based refinement. Trade-offs like these were essential to make the project feasible within available resources and to ensure repeatability across objects.

A.1.4 Preprocessing Pipelines

To ensure compatibility with FoundationPose, all mesh outputs were post-processed through a standard alignment and formatting pipeline. This included:

- **Scaling and centering:** All meshes were normalized to fit within a canonical bounding box centered at the origin.
- **Coordinate system alignment:** Meshes were aligned to match the camera-frame conventions used in FoundationPose.
- **Texture validation:** OBJ, MTL, and texture files were checked for file integrity, UV consistency, and loadability by the rendering engine.
- **Format conversion:** Non-standard exports were converted into OBJ+MTL+PNG triplets using `threestudio`'s mesh-exporter and a custom shell script.

These steps were essential to ensure that all generated models could be reliably passed into FoundationPose without manual intervention or adjustment.

A.2 FoundationPose System Adaptations: RTX 4070, CUDA 12.1, and Compiler Compatibility

To ensure that FoundationPose could be executed reliably on modern hardware, several system-level adaptations were necessary. The official Docker environment provided with the framework was originally configured for older NVIDIA driver stacks and CUDA versions (specifically CUDA 11.x), which posed compatibility issues when attempting to run the pipeline on our local machine equipped with an RTX 4070 GPU.

To address these issues and achieve stable performance and reproducibility, we introduced the following modifications:

- **Base Image Upgrade:** The Docker base image was replaced with `nvidia/cuda:12.1.0-devel-ubuntu20.04`, which supports the newer CUDA 12.1 toolkit and is compatible with the RTX 4070's driver requirements.
- **PyTorch Upgrade:** The framework's deep learning components were migrated to `PyTorch 2.1.0+cu121`, ensuring full compatibility with CUDA 12.1 while maintaining consistency with the latest stable APIs.
- **nvdiffrast Rebuild:** NVIDIA's differentiable rasterizer, `nvdiffrast`, was rebuilt from source to target CUDA 12.1, as precompiled binaries were incompatible with newer driver architectures.
- **pytorch3d Patch:** The `pytorch3d` library, responsible for mesh processing and rendering, was patched to resolve compatibility issues that arose with updated dependencies and newer compute capabilities.
- **Compiler Flags Adjustment:** The C++ standard used during compilation was raised from `-std=c++14` to `-std=c++17` to support modern versions of `libc++` and address compatibility with newer toolchains.

These adaptations were crucial in enabling a stable and fully functional runtime environment for FoundationPose on contemporary GPUs. They also ensured that our experiments remained reproducible and compatible across modern systems, particularly when integrating with custom pipelines or deploying to other high-performance platforms.

A.3 FoundationPose Model-Free Pipeline Details

While the model-free mode, described in the official GitHub repository and supporting literature, offers an alternative path through NOFs, it requires training a new neural field for each object using a small number of reference RGB-D images. This process includes several stages: preprocessing the views, generating signed distance function (SDF) representations, running scripts such as `bundlesdf/run_nerf.py` to optimize the NOF, and finally using the resulting field to synthesize inputs for downstream pose estimation.

Despite its potential, we identified three key limitations that render the model-free pipeline unsuitable for our purposes:

1. **Lack of Clear Integration and Documentation.** The instructions for the model-free setup are fragmented across optional scripts, not integrated into the primary code paths (`run_linemod.py` or `predict_pose_refine.py`), and lack pretrained NOFs. The documentation also omits important details on expected runtime and performance, making reproducibility in a standardized setup difficult.

2. **Need for Object-Specific Training.** Even though NOFs are lighter than retraining the full pose network, the requirement to train a new model for each object contradicts our goal of evaluating general-purpose pipelines. By contrast, GenAI-based mesh reconstruction methods like Magic123 and Zero123 allow us to generate usable 3D models without any fine-tuning or training.

3. **No Practical Advantage in Our Setup.** Functionally, the model-free pipeline synthesizes views that are processed through the same refinement and scoring components used in the model-based pipeline. Since pretrained generative models can directly produce 3D meshes from a single RGB image at test time, without per-object optimization, they align more naturally with our objective: assessing robustness and generalization in a zero-shot setup.

Accordingly, our experimental framework relies on the `linemod` (approximately 8 GB) and `demo_data` pipelines for both integration and evaluation. The `YCB` pipeline, although supported by FoundationPose, was excluded primarily due to the large size of the underlying dataset (approximately 200 GB). Since our focus is on single-frame prediction and generalization analysis, the substantial storage and preprocessing overhead of YCB made it impractical within the constraints of this study.

A.4 FoundationPose Model-Based Pipeline Inputs

To perform 6D object pose estimation, FoundationPose relies on a structured set of inputs that collectively capture both the appearance and spatial structure of the scene. These inputs are critical for both the rendering-based hypothesis generation and the downstream refinement and scoring stages. Below, we outline each required (and optional) input component and its role in the pipeline.

1. RGB Image **Purpose:** Provides the visual appearance of the object, including texture, color, and contours.
Explanation: The RGB image serves as the main modality for appearance-based comparison between the real scene and synthetic renderings. Both the refiner and scorer networks extract features from the RGB input to evaluate how well a rendered pose hypothesis aligns with the observed image. Texture information is particularly crucial for distinguishing between visually similar poses, such as the front versus back of a textured object.

2. Depth Map **Purpose:** Enables computation of the scene's 3D geometry by capturing the distance of each pixel from the camera.
Explanation: The depth map is converted into a 3D point cloud or xyz-map using the camera intrinsics. This geometric representation provides spatial context to the visual features and is essential for generating input tensors such as surface normals or 3D coordinates. These geometric signals enhance pose estimation, particularly in scenarios involving occlusions or textureless surfaces.

3. Camera Intrinsics (K) **Purpose:** Maps pixel coordinates to 3D rays and vice versa, enabling projections between image and world space.
Explanation: The camera intrinsics, typically represented as a 3×3 matrix, are used in multiple stages of the pipeline:

- Projecting 3D mesh points into 2D image space for rendering.
- Lifting depth maps into 3D point clouds.
- Ensuring spatial consistency during cropping and resizing.

Accurate intrinsics are vital; any deviation leads to incorrect rendering and geometric misalignment.

4. 3D Mesh of the Object **Purpose:** Enables rendering of synthetic RGB, depth, and normal maps from multiple candidate poses.
Explanation: In the model-based pipeline, the 3D mesh serves as the geometric proxy for rendering synthetic observations. Using a differentiable renderer (e.g., `nvdiffrast`), the mesh is rendered under different candidate poses and compared with real input data. This comparison forms the basis for scoring and refining pose hypotheses. For optimal results, the mesh should be watertight, correctly scaled, and centered. In our thesis, this input is mandatory since the model-free NOF setup is not used.

5. (Optional) Object Mask **Purpose:** Focuses rendering and comparison on the relevant object region, improving accuracy and reducing noise.
Explanation: Although optional, the object mask improves several steps in the pipeline:

- Removes background pixels from consideration.
- Enhances crop alignment for both input and synthetic views.
- Reduces sensitivity to occlusions and nearby distractors.

When a mask is unavailable, FoundationPose assumes a single-object scene or uses previous tracking information to localize the object, which is often sufficient for benchmarks like LINEMOD.

6. (Optional) Symmetry Information **Purpose:** Handles symmetric objects correctly during scoring by avoiding penalties for visually indistinct but geometrically different poses.
Explanation: Some objects exhibit symmetry under specific transformations such as soda cans, scissors, or bowls. FoundationPose provides architectural support for symmetry-aware scoring using symmetry transformations (`symmetry_tfs`) during the candidate evaluation stage. However, in practice, this mechanism is underutilized:

- `symmetry_tfs` is never populated by default.
- No parsing logic exists for extracting symmetry from `models_info.yml`.
- No built-in loss or scoring adjustments apply symmetry unless provided manually.

In our evaluation, the LINEMOD subset was chosen partly because it contains primarily asymmetric objects, simplifying the analysis. However, the HOTS dataset includes several highly symmetric objects, such as Pringles tubes, soda cans, and scissors, yet provides no associated symmetry metadata. As a result, FoundationPose treats these poses literally, leading to potentially inflated error metrics. We acknowledge this as a limitation and propose that future work explore symmetry-aware scoring, particularly when applying FoundationPose to general-purpose datasets where symmetry is common but metadata is absent.

A.5 Step-by-Step FoundationPose Pipeline

The FoundationPose framework follows a structured, multi-stage pipeline to perform 6D pose estimation. Each stage transforms raw input into increasingly refined predictions, culminating in the selection of the most confident pose hypothesis. The process is designed to balance geometric accuracy and appearance-based alignment through a combination of differentiable rendering, deep feature extraction, and transformer-based learning. The complete pipeline is outlined below.

Step 1: Load and Preprocess Input FoundationPose begins by loading all available input data using its modular reader components:

- Loads the RGB image, depth map, camera intrinsics, and object mask (if available).
- Converts the depth map into a 3D point cloud or xyz-map using intrinsics.
- Resizes, centers, and scales the 3D mesh to fit the canonical space.
- Voxelizes the point cloud for efficient GPU-based computation.

These steps establish the spatial and visual context required for downstream rendering and inference.

Step 2: Pose Hypothesis Sampling Next, the pipeline generates a diverse set of coarse pose hypotheses:

- Virtual cameras are placed uniformly over an icosphere, typically yielding 40 or more viewpoints.
- Each view is combined with in-plane image rotations (e.g., $0°$ to $360°$ in $60°$ steps).
- Transformations are inverted to simulate object-in-camera coordinates.

Each resulting hypothesis serves as a candidate pose that will be evaluated against the input image.

Step 3: Rendering (nvdiffrast) Each candidate pose is rendered using `nvdiffrast`, a high-performance differentiable GPU rasterizer:

- Produces synthetic RGB, depth, and normal maps for each hypothesis.
- Computes per-pixel 3D coordinates (xyz-map) from the rendered geometry.
- Applies learned crop and alignment transformations to match synthetic and real data spatially.

This stage bridges the visual domain between real observations and synthetic hypotheses.

Step 4: Pose Refinement (Refiner Network) The PoseRefinePredictor module compares each rendered view to the actual scene and predicts pose corrections:

- Two CNN encoders (`encodeA`) extract features from the real and rendered images.
- Feature maps are fused using a residual ResNet backbone (`encodeAB`).
- Positional embeddings are added, and two Transformer heads predict:
  - ΔR: a rotation update (as either axis-angle or 6D representation).
  - Δt: a translation vector update.

Typically, 1 to 3 refinement iterations are performed to progressively improve pose accuracy.

Step 5: Scoring and Selection (Scorer Network) The final stage scores each refined hypothesis and selects the most confident one:

- A CNN encoder processes each rendered–real image pair to extract features.
- Multi-head self-attention layers model the relationships between all candidate poses.
- Cross-attention mechanisms aggregate global context to weigh each hypothesis.
- A linear prediction head assigns a confidence score to each candidate.

The pose with the highest score is returned as the final output.

This sequential pipeline allows FoundationPose to combine coarse pose sampling, learned refinement, and global scoring into a coherent, end-to-end estimation system. Its modular structure also facilitates evaluation and substitution of individual components, making it suitable for research on both generalization and robustness.

A.6 FoundationPose Usage Modes

A.6.1 The Three Usage Modes of FoundationPose

In this thesis, we focused primarily on the `demo_data` and `linemod` pipelines. The `demo_data` pipeline enabled rapid qualitative testing and integration of reconstructed meshes, while the `linemod` pipeline served as the backbone for our quantitative benchmarking. The `YCB` pipeline, although supported, was excluded due to its storage demands and lack of alignment with our single-frame evaluation focus.

4.1 The `demo_data` Pipeline The `demo_data` pipeline is a lightweight setup designed for single-frame predictions with a known mesh. It is executed via the `run_demo.py` script, which internally invokes core components such as `predict_pose_refine.py` and `predict_score.py`.

Purpose:

- Rapid testing of the FoundationPose pipeline, GPU compatibility, and rendering functionality.
- Debugging the refiner and scorer components in isolation, on a per-frame basis.
- Performing qualitative evaluations of pose predictions using reconstructed meshes from GenAI models such as Magic123 or Zero123.

Characteristics:

- Operates on a single object at a time (typically a mustard bottle example).
- Uses a YCB-style input format via the `YcbineoatReader`, though it does not require the full YCB-Video dataset.
- Accepts RGB-D input, camera intrinsics, and a mesh as input.
- Supports both refiner and scorer networks in combination.
- Enables hypothesis generation, refinement, and scoring for a single frame.

Outputs:

- Refined and scored 6D pose predictions.
- Visualizations with overlaid RGB images showing the predicted pose.
- Optional logs and saved inference outputs.

While not designed for benchmarking accuracy, this mode was essential for integration testing and validating the compatibility of custom, generated meshes with FoundationPose.

4.2 The Full `linemod` Pipeline The `linemod` pipeline is the standard evaluation workflow used in FoundationPose, tailored to the BOP dataset format. It is executed via `run_linemod.py` and supports large-scale, multi-object evaluation across hundreds of frames.

Purpose:

- Conducting systematic, quantitative evaluation on the LINEMOD benchmark.
- Measuring the accuracy and consistency of the refiner and scorer modules across multiple test samples.
- Comparing the performance of FoundationPose using either ground-truth meshes or AI-generated alternatives.

Characteristics:

- Requires the full LINEMOD dataset in BOP format, including RGB-D frames, object masks, intrinsics, and CAD models.
- Uses the `BopBaseReader` for standardized input parsing and batch preparation.
- Runs batch-based hypothesis sampling, rendering, refinement, and scoring over all test frames.

Outputs:

- Final 6D pose predictions for each frame and object.
- Logs pose estimations and optionally saves visualizations for each frame.
- Quantitative evaluation metrics such as ADD-S.

This pipeline formed the backbone of our main experiments, allowing for a direct comparison between pose predictions derived from original LINEMOD meshes and those reconstructed using GenAI methods.

4.3 The `YCB` Pipeline (Excluded) FoundationPose also supports the YCB-Video dataset, a large-scale benchmark (approximately 200 GB) designed for real-world 6D object pose tracking over video sequences. Although the pipeline offers support for complex multi-object and multi-frame scenarios:

- It was excluded from this thesis due to the substantial size of the YCB dataset and the associated storage and preprocessing requirements.
- Its tracking-oriented nature does not align with our focus on single-frame generalization and evaluation using plug-and-play reconstructed meshes.

Notably, the `demo_data` pipeline reuses components from the YCB pipeline, particularly the `YcbineoatReader`, but applies them in a simplified, single-frame context. This made it significantly more manageable for the goals of this study.

A.7 ThreeStudio Mesh Export Details

Figure A.1 shows how varying isosurface threshold and resolution affects mesh completeness and noise.

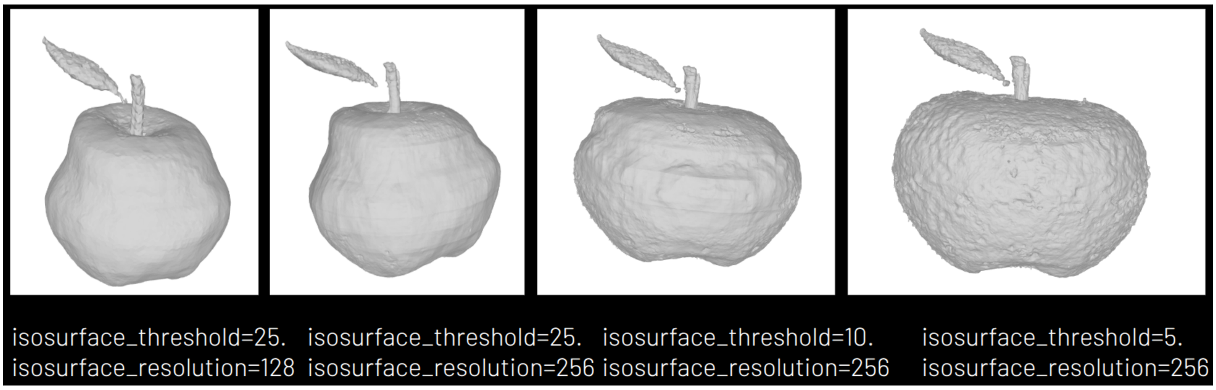


Figure A.1: Effect of different isosurface threshold and resolution settings on exported mesh quality. From left to right: smoother but incomplete mesh (threshold=25, resolution=128) to noisy but highly detailed geometry (threshold=5, resolution=256).

Export Mechanism We activate mesh export via ThreeStudio's `-export` flag in `launch.py`, producing textured OBJ+MTL (with UV maps) or OBJ with vertex colors. Key exporter parameters include:

- `system.exporter.fmt=obj-mtl`
- `system.geometry.isosurface_method=mc-cpu`
- `isosurface_resolution=256`
- `isosurface_threshold=5.0`
- `save_uv=True`, `save_texture=True`, `texture_format=png`

Custom Batch Script We wrote a shell wrapper that:

1. Recurses through each model's output folder.
2. Loads the final checkpoint and config.
3. Calls the exporter with our standard flags.

Parameter Justification Default exports (threshold=25, resolution=128) often left holes; lowering the threshold and upping the resolution recovered thin features (e.g. drill bits) without excessive noise. Figure A.1 illustrates these trade-offs.

This controlled export pipeline ensured that all meshes, regardless of their generative origin, were produced with comparable structure, texture fidelity, and scale, making them suitable for direct evaluation within the FoundationPose system.

A.8 Threestudio Zero123 Details

Table A.1: Comparison of the original Zero123 and the `threestudio`-adapted variant used in this thesis.

Component	Original Zero123	`threestudio` Variant
Output	Novel 2D images	Gradients for SDS (no image output)
Usage Mode	Image synthesis	View-consistency regularization
Training Loop	Fully generative	Backpropagation-based 3D optimization
Prompt/Text Conditioning	No	No
VRAM Requirement	24–32 GB	12–32 GB

Implementation via `threestudio`. Instead of using the original Zero123 implementation as a standalone view generator, we adopted the `threestudio` variant, which integrates Zero123 into a Score Distillation Sampling (SDS) loop. In this setting, Zero123 serves as a teacher model or "view-consistency critic," enforcing geometric plausibility during 3D optimization.

The differences between the original Zero123 implementation and the `threestudio`-adapted variant used in this thesis are summarized in Table A.1. These changes reflect a shift from image generation to a supervisory role within a gradient-based optimization loop.

Technical Process. In the `threestudio` workflow, Zero123 is used as part of an SDS setup tailored for camera-aware supervision. During each training iteration, a set of virtual camera poses is sampled, and the current 3D model is rendered from those views. Zero123 evaluates the consistency of these renderings relative to the single input image (from the HOTS dataset), and its feedback is used to compute gradients that refine the 3D shape. Importantly, Zero123 operates only in the image domain and does not directly supervise texture or mesh detail.

This setup avoids the complexity of manually stitching multi-view predictions, while also reducing training time and memory usage. Training typically converged in under 15 minutes for 600 iterations on a 32 GB GPU.

Advantages and Trade-Offs. The integration of Zero123 offered several advantages:

- Efficient memory use and short training times, even on mid-range hardware.
- High-quality geometric supervision with minimal configuration overhead.
- No need for explicit image synthesis or multi-view aggregation.

However, certain trade-offs were observed:

- Zero123 does not improve visual realism, it focuses solely on geometry.
- No visual output is produced directly by the model.
- The SDS loss is sensitive to initial pose misalignments during early training.

Despite these limitations, Zero123 proved highly effective as a lightweight supervisory signal for early-stage 3D reconstruction. Its pose-awareness and low integration overhead made it a key component of the single-image-to-3D pipeline used throughout this thesis.

Table A.2: Comparison of the official Magic123 implementation and the `threestudio` variant used in this thesis.

Component	Official Magic123	`threestudio` Variant (Used Here)
Geometry Optimization	SDS with Zero123 guidance	SDS with Stable Diffusion v1.5
Mesh Refinement	3D Diffusion with texture synthesis	Omitted (coarse stage only)
View Conditioning	Yes (camera pose input)	Yes (Zero123-style)
Input Modalities	Image + Text Prompt	Image + Text Prompt
Output	UV-mapped OBJ mesh	OBJ mesh
Minimum VRAM	24–40 GB	20 GB

A.9 Threestudio Magic123 Details

Implementation via `threestudio`. Due to high memory demands and long runtimes, we did not use the official Magic123 pipeline. Instead, we adopted the `threestudio` variant, which simplifies the second stage and runs efficiently on consumer GPUs.

In this adaptation, the coarse mesh is generated via Score Distillation Sampling (SDS), guided by Stable Diffusion v1.5 using both the input image and a descriptive text prompt. The 3D diffusion-based refinement stage is skipped entirely. Despite this simplification, the generated meshes were structurally sound and suitable for use in FoundationPose.

This hybrid setup, combining SDS with pose-aware supervision, follows a DreamFusion-style approach but replaces proprietary components like Imagen with open-source Stable Diffusion. It enables efficient mesh reconstruction without compromising usability in downstream tasks.

Technical Process. Magic123 operates in two stages:

- **Stage I – Coarse Optimization:** SDS optimizes a volumetric representation into a coarse mesh using 2D feedback from Stable Diffusion, guided by both an RGB image and a text prompt.
- **Stage II – (Skipped):** The full 3D diffusion refinement phase is omitted in our experiments to avoid texture distortion and excessive VRAM consumption.

Rendered views from the current mesh are evaluated by the SDS model, which provides gradient feedback to update geometry. Pose conditioning ensures consistency with the input image's viewpoint.

Advantages and Trade-Offs. The simplified Magic123 pipeline used in this thesis provided a practical balance of speed, memory, and mesh quality:

- Accepts both an RGB image and text prompt, allowing richer scene interpretation
- Requires only 10–12 GB of VRAM
- Avoids training instabilities and distortions observed in symmetric objects
- Produces OBJ-format meshes with sufficient detail and UV-mapped textures
- Minor reductions in texture realism were acceptable given the task focus

These trade-offs make Magic123 an ideal candidate for reconstructing general-purpose meshes in resource-constrained or real-world scenarios where object CAD models are unavailable.

A.10 Threestudio Dreamfusion Details

Implementation in `threestudio` Using DeepFloyd IF. Because the original DreamFusion uses Imagen, which is not publicly available, this thesis implements the method using the `threestudio` framework, with DeepFloyd IF as the guiding diffusion model. Compared to alternatives like Stable Diffusion, DeepFloyd IF provided:

- More coherent geometry across object parts
- Better adherence to textual concepts
- Consistent surface structure in visually complex objects

The training pipeline used `dreamfusion-if.yaml` as the base config, requiring approximately 15 GB VRAM for text embedding and 10 GB for training. Prompt examples included objects like "a keyboard," "a can of soda," and "a pair of scissors," matching the real-world object categories tested in HOTS and LINEMOD.

DreamFusion as an Architectural Blueprint. While DreamFusion is used here as a standalone model, its broader architectural influence extends to the entire pipeline used in this thesis. All three generative methods, Zero123, Magic123, and DreamFusion, share a common foundation in DreamFusion's SDS logic:

- **DreamFusion:** SDS + text-only guidance
- **Magic123:** SDS + image and text guidance
- **Zero123:** SDS + image-to-view consistency

Each model distills 2D guidance into 3D structure, with no need for ground-truth 3D data. DreamFusion is thus not only a standalone baseline, but the conceptual and algorithmic core underlying all generation pipelines in this thesis.

Technical Process. The DreamFusion workflow consists of:

- Feeding a text prompt (e.g., "a hamburger") into the system
- Initializing a random 3D volume (e.g., SDF or NeRF)
- Rendering the volume from a camera pose
- Passing the rendered image through the 2D diffusion model's denoiser
- Using SDS gradients to update the 3D geometry

This loop repeats until the 3D object appears consistent with the prompt across viewpoints, despite never seeing any 3D ground truth.

Observations and Trade-Offs. DreamFusion provided valuable insight into the text-to-3D generation landscape and its limitations:

Advantages:

- Fully self-supervised, requires only a text prompt
- Strong performance with vivid, classifiable prompts
- Useful in scenarios where no visual or CAD input is available

Trade-Offs:

- Geometry was often less detailed than Magic123
- Prompt specificity significantly affected output quality
- Failed more frequently on symmetric or fine-structured objects

Despite these limitations, DreamFusion was a key part of this thesis as both a baseline and a conceptual foundation. It demonstrated the potential of pure text-to-3D generation while also highlighting the need for richer supervision when geometric precision is required for tasks like 6D pose estimation.

A.11 Pipeline Diagrams

Below are the high-level sequence and flow diagrams for each of our four thesis pipelines:

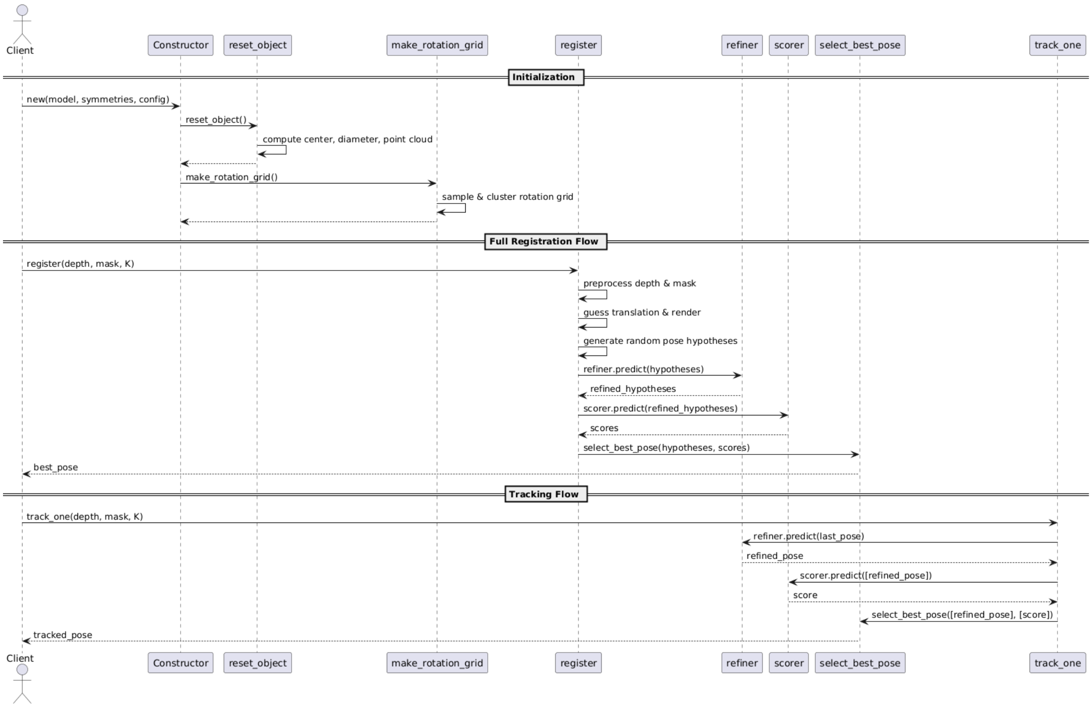


Figure A.2: FoundationPose full tracking sequence: initialization (mesh centering, symmetry grid), full registration (depth+mask preprocessing, hypothesis generation, refinement, scoring), and tracking (single-hypothesis refine+score loop).

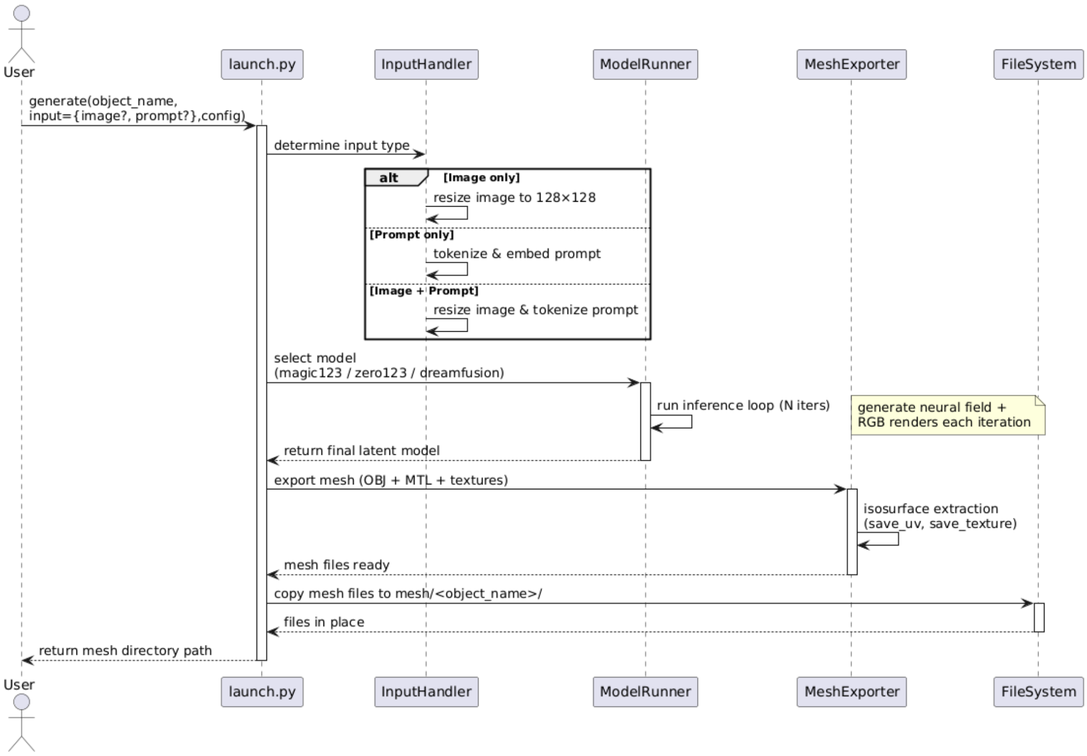


Figure A.3: ThreeStudio MeshGen pipeline: input handling (image vs. prompt), model selection (Magic123, Zero123, DreamFusion), neural field inference loop, isosurface extraction, and mesh export (OBJ+MTL+textures).

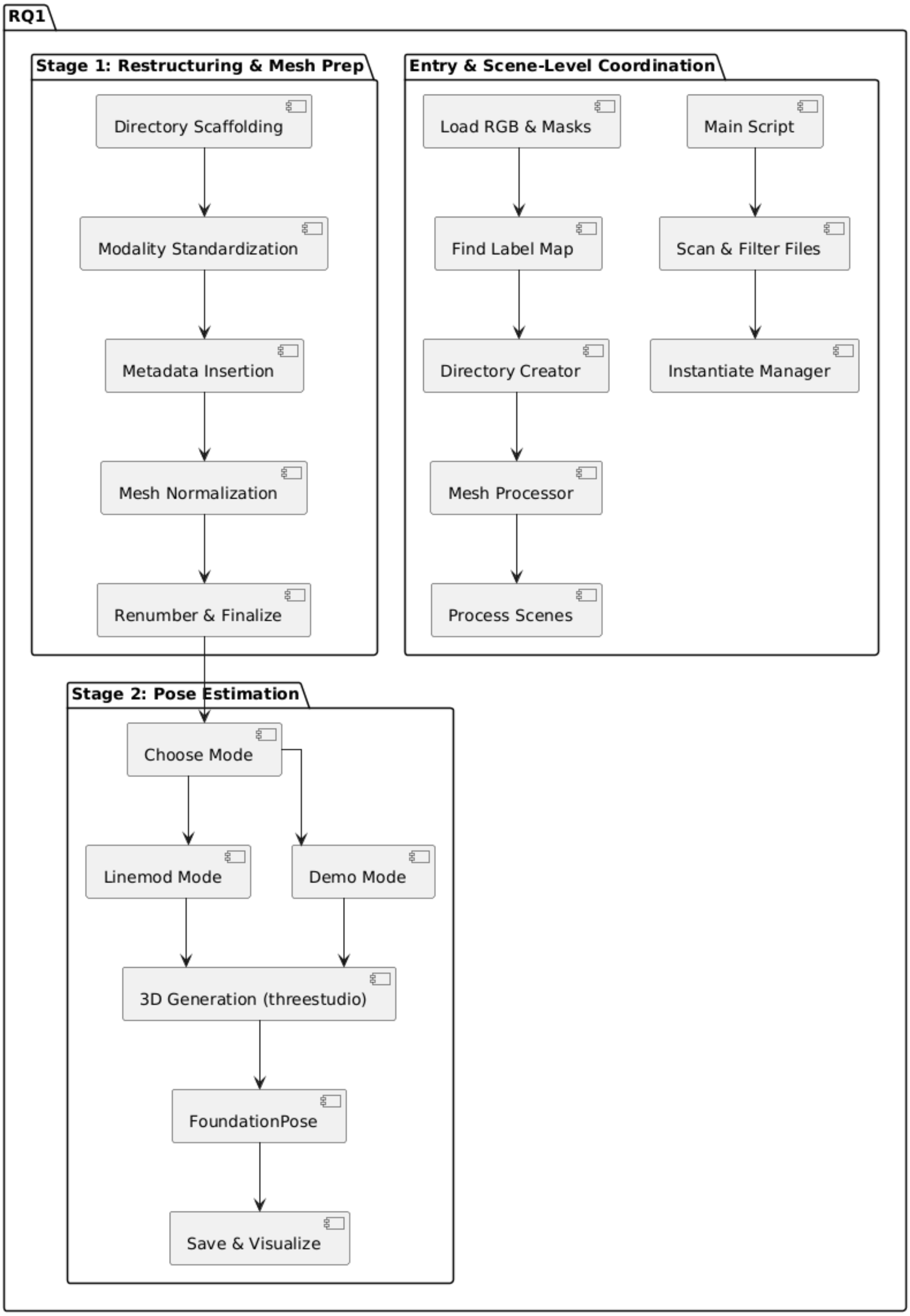


Figure A.4: RQ1 pipeline: restructuring & mesh preparation (directory scaffolding, modality standardization, metadata insertion, mesh normalization), followed by pose estimation (Threestudio 3D generation in Linemod or demo mode, then FoundationPose, save & visualize).

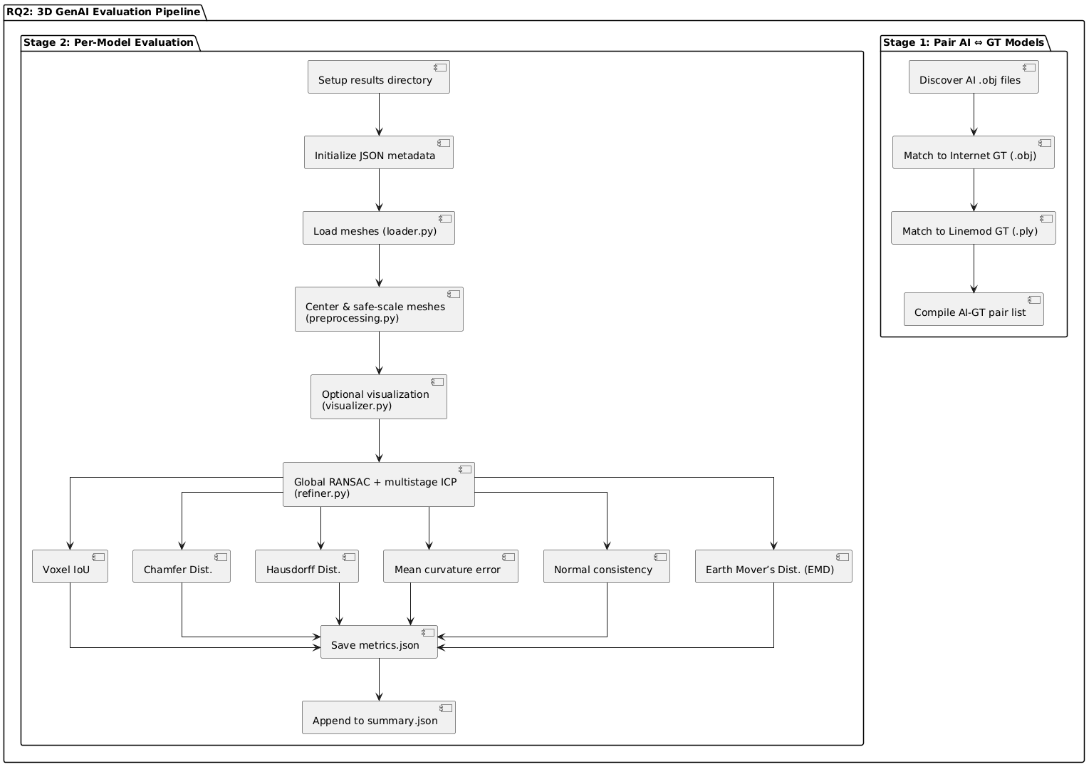


Figure A.5: RQ2 pipeline: pairing each AI-generated mesh with its CAD ground truth, preprocessing (centroid normalization, safe scaling, RANSAC, ICP), and computing six fidelity metrics (Voxel IoU, Chamfer, Hausdorff, MCE, Normal Consistency, EMD) per model and time budget.

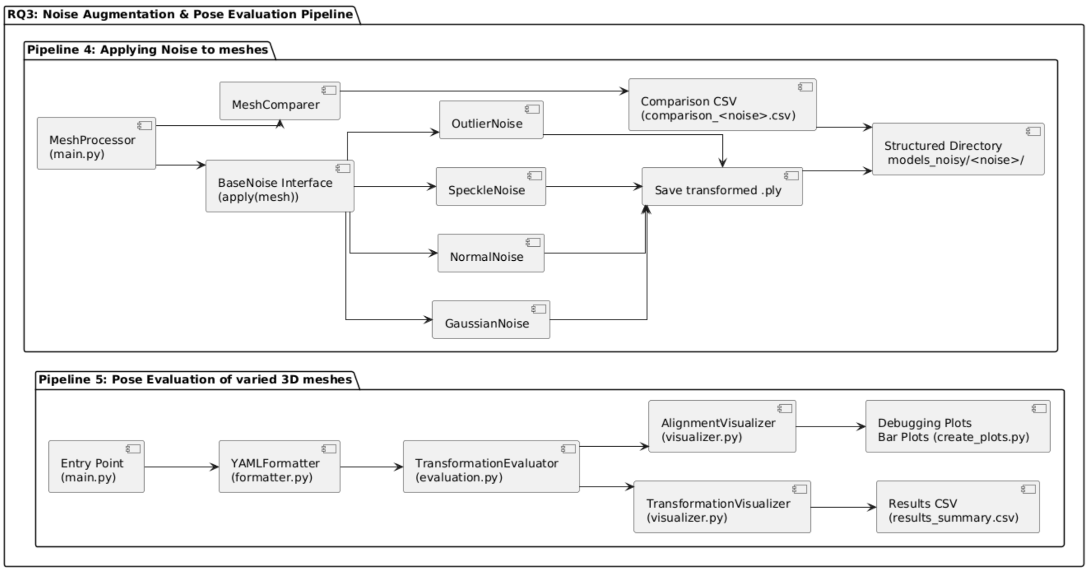


Figure A.6: RQ3 pipeline: noise augmentation of CAD meshes (outlier, speckle, normal, gaussian), followed by pose-estimation evaluation under occlusion and clutter (alignment visualizer, debugging plots, results summarization).

A.12 Config Files

Detailed configuration parameters for the RQ1 experiments on HOTS.

RQ1: HOTS Pipelines

1. HOTS Restructuring Pipeline

- **FORMAT_TYPE = demo**: flattened per-object layout (no subfolders).
- **REQUIRE_DEPTH_NPY = True**: only '.npy' depths accepted.
- **TARGET_DIMS:**

```
{ apple:0.08, banana:0.15, book:0.22, bowl:0.19, can:0.12,
cup:0.11, fork:0.19, juice_box:0.17, keyboard:0.45, knife:0.20,
laptop:0.33, lemon:0.08, marker:0.15, milk:0.24, monitor:0.33,
mouse:0.11, orange:0.08, peach:0.08, pear:0.08, pen:0.15,
plate:0.24, pringles:0.23, scissors:0.17, spoon:0.19, stapler:0.18 }
default = 0.1 m
```

- **SHARED_CATEGORIES:**

```
{ book:Book, can:Can, cup:Cup, fork:Fork, marker:Marker,
pen:Pen, plate:Plate, pringles:Pringles, scissors:Scissors, spoon:Spoon
}
```

- **ROTATION_X/Y/Z = 0,0,0**: no axis alignment.

2. HOTS Pose Estimation Pipeline

- **PIPELINE_MODE = demo**: interactive mode.
- **PROCESS_ALL_OBJECTS = True**
- **USE_MASK_EVERY_FRAME = True**: full registration.
- **ITERATION_REGISTER / ITERATION_TRACK = 5 / 2**
- **DEBUG_LEVEL = 2**: intermediate debug outputs.
- **AXIS_SCALE / AXIS_THICKNESS / TRANSPARENCY = 0.1/3/0**: overlay styling.
- **SKIP_FRAMES_CONTAINING = ["kitchen"]**: omit frames with "kitchen."
- **DEVICE = cuda:0**: use GPU.

RQ2

Directory & Sampling Parameters

- **VOXEL_SIZE** = 7.0 mm: default voxel size for point-cloud operations.
- **DEFAULT_NUM_SAMPLES** = 30 000: standard point-cloud sampling density.
- **EMD_NUM_SAMPLES** = 2 000: points per object for Earth Mover's Distance.

Metric Thresholds

- **IOU_THRESHOLDS**:

  `{ excellent: 0.90, good: 0.75, warning: 0.50, bad: 0.30, critical: 0.10 }`

- **CHAMFER_THRESHOLDS**:

  `{ excellent: 0.02, good: 0.05, warning: 0.10, bad: 0.20, critical: 0.30 }`

- **HAUSDORFF_THRESHOLDS**:

  `{ excellent: 0.01, good: 0.05, warning: 0.10, bad: 0.20, critical: 0.30 }`

- **EMD_THRESHOLDS**:

  `{ excellent: 0.01, good: 0.05, warning: 0.10, bad: 0.20, critical: 0.30 }`

- **NORMAL_CONSISTENCY_THRESHOLDS**:

  `{ excellent: 0.95, good: 0.85, warning: 0.70, bad: 0.50, critical: 0.30 }`

- **MEAN_CURVATURE_THRESHOLDS**:

  `{ excellent: 0.005, good: 0.010, warning: 0.020, bad: 0.040, critical: 0.080 }`

RQ3

Noise Generation The following noise types (from `config.py`) are used to perturb each CAD mesh:

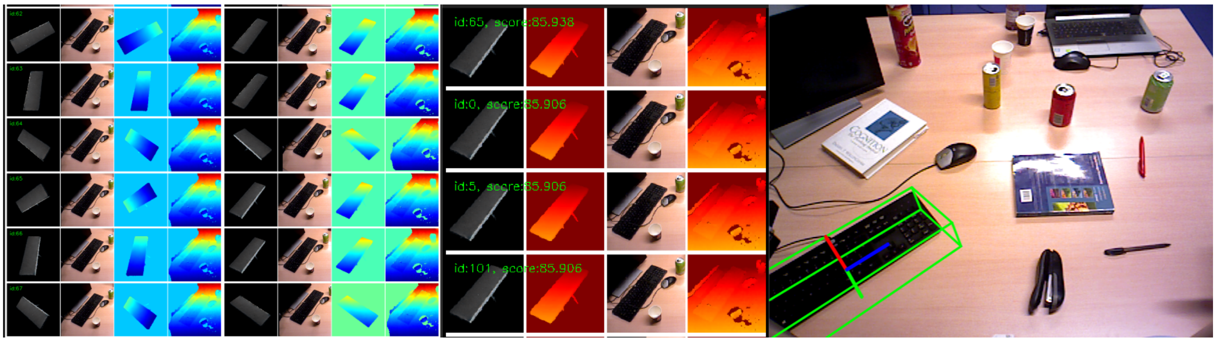
- **gaussian:** `GaussianNoise(mean=0, std_dev=0.1)`, mild uniform jitter.
- **normal:** `NormalNoise(std_dev=0.1)`, displacements along each vertex's surface normal.
- **speckle:** `SpeckleNoise(std_dev=0.1)`, multiplicative scaling of each point's radius.
- **outlier:** `OutlierNoise(percentage=0.02, std_dev=0.1)`, sparse, large random jumps at 2

Pose Evaluation All variants (original GT, the four noise types, plus Zero123, Magic123, DreamFusion) are passed through the same pose-estimation routine. Key config parameters:

- **LINEMOD_ROOT** = `"linemod_results"`
- **OBJECT_IDS** = [1, 4, 6, 9, 10]
- **POSE_METHODS** = [original, normal, gaussian, outlier, speckle, zero123, magic123, dreamfusion]
- **FRAME_IDX** = 0
- Zoom factors: **FULL_ZOOM_FACTOR**=0.4, **GIF_ZOOM_FACTOR**=0.5
- **TREND_THRESHOLDS** for reporting: { rotation: [5.0, 10.0]°, translation: [0.02, 0.05] m, pose: [0.05, 0.13] (Frobenius), add: [0.02, 0.05] m }
- **LABELS** = { ("Rotation Error","°","blue","rotation"), ("Translation Error","m","orange","translation"), ("Pose Error","","green","pose"), ("ADD Error","m","purple","add") }

A.13 RQ1: Additional Examples

(a) ***Scissors*** **object: sampled poses, depth refinement, scorer heatmap, final overlay.**



(b) ***Keyboard*** **object: sampled poses, depth refinement, scorer heatmap, final overlay.**

Figure A.7: Visualizations of FoundationPose's pipeline stages for two representative objects stacked vertically. Each subfigure shows, from left to right, random pose hypotheses, refined depth alignment, scorer confidence heatmap, and final pose overlay on the RGB frame.

A.14 RQ2: Generative-AI Reconstruction Prompts and Inputs

DreamFusion-IF Prompts The following single-view textual descriptions were used to generate each object's mesh with DreamFusion-IF:

- "Banana. A curved yellow banana with a smooth peel."
- "Camera. A DSLR camera with a black and silver body, a large zoom lens with a ridged focus ring. It has a right-hand grip, a shutter button on top, and a visible hot shoe mount. The surface is mostly matte plastic."
- "Cat. A small pink cat figurine with a large rounded head, short legs, and a raised tail. The cat has small black eyes, a tiny nose, and is in a walking pose."
- "Duck. A small yellow rubber duck with a round body, a slightly raised tail, and molded wings. It has large circular eyes and an orange beak that is slightly open."
- "Gorilla. A stylized seated gorilla figurine with short, thick limbs, a rounded body, and a smooth head. The figure is red, has minimal facial features, and is leaning slightly forward."
- "Mouse. A wireless mouse with a slightly domed top and a sleek, ergonomic shape. It has a central scroll wheel and two click buttons molded into a seamless top shell."
- "Soda Can. A standard aluminum soda can with a cylindrical shape. The surface features bright orange branding, with minimal bumps or texture variation. The form is vertically symmetrical."

Magic123 Prompts For Magic123 we used concise, object-centric labels:

- "Banana (Curved Yellow Banana)"

- "Camera (DSLR Camera)"
- "Pink Cat (Toy Cat Figurine)"
- "Yellow Duck (Rubber Duck)"
- "Red Gorilla (Gorilla Figurine)"
- "Mouse (Wireless Computer Mouse)"
- "Soda Can (Orange Soda Can)"

GenAI Input Images Figure A.8 shows the full set of input images used by Magic123 and Zero123

Figure A.8: Representative input images used by Magic123 and Zero123 for 3D reconstruction. Each image was originally at 1024×1024 px, then automatically downsampled by ThreeStudio's preprocessing to 128×128 px prior to mesh synthesis.

A.15 RQ2: 3D Reconstruction Examples

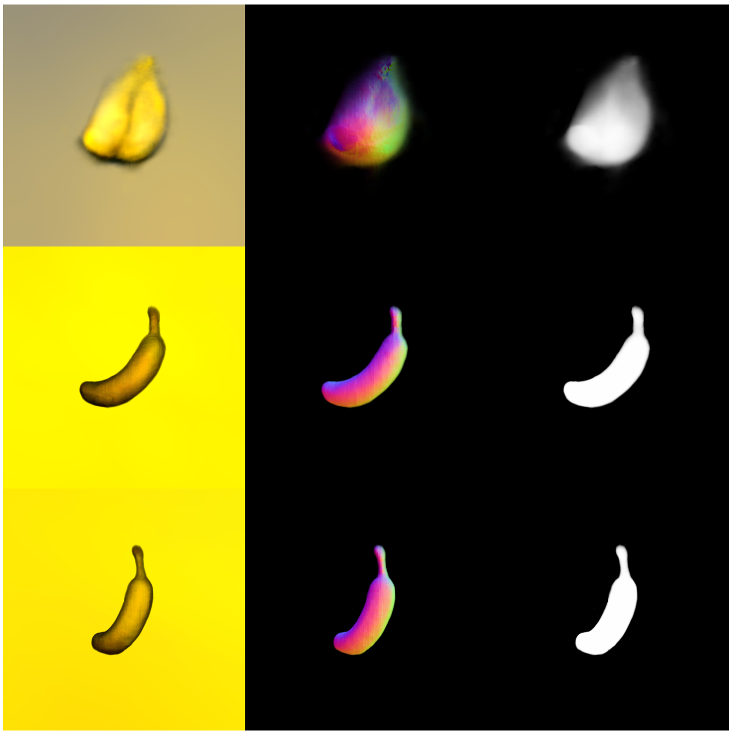


Figure A.9: DreamFusion reconstructions of the banana prompt after 10 min (top), 30 min (middle), and 60 min (bottom) of synthesis.

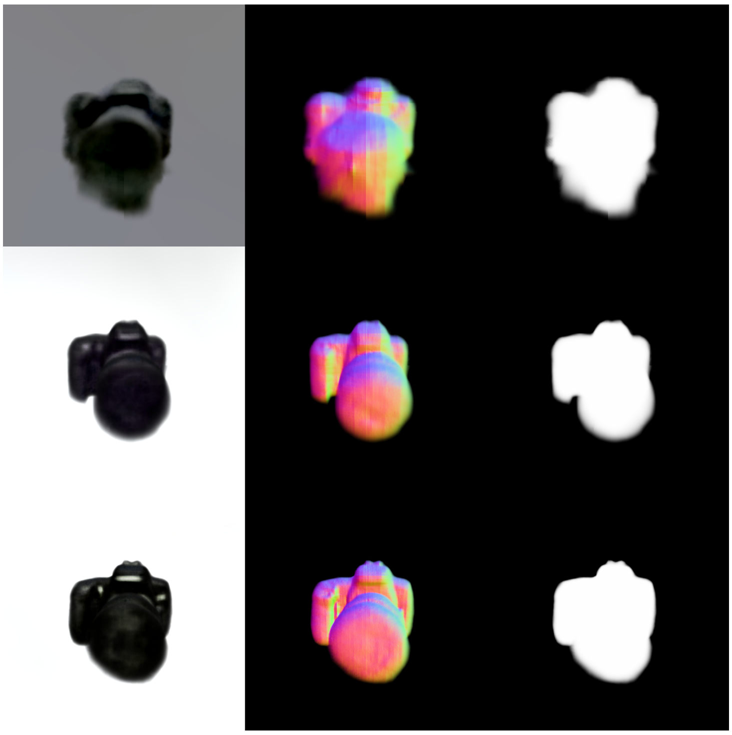


Figure A.10: DreamFusion reconstructions of the camera prompt after 10 min (top), 30 min (middle), and 60 min (bottom) of synthesis.

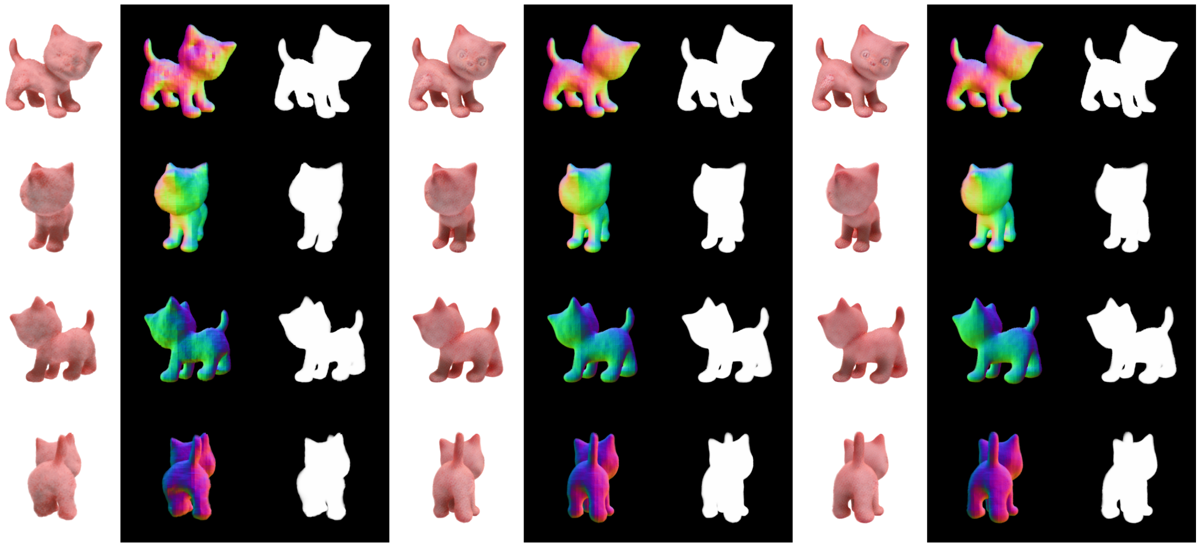


Figure A.11: Magic123 reconstructions of the cat after 10 min (left 3 columns), 30 min (middle 3 columns), and 60 min (right 3 columns) of synthesis.

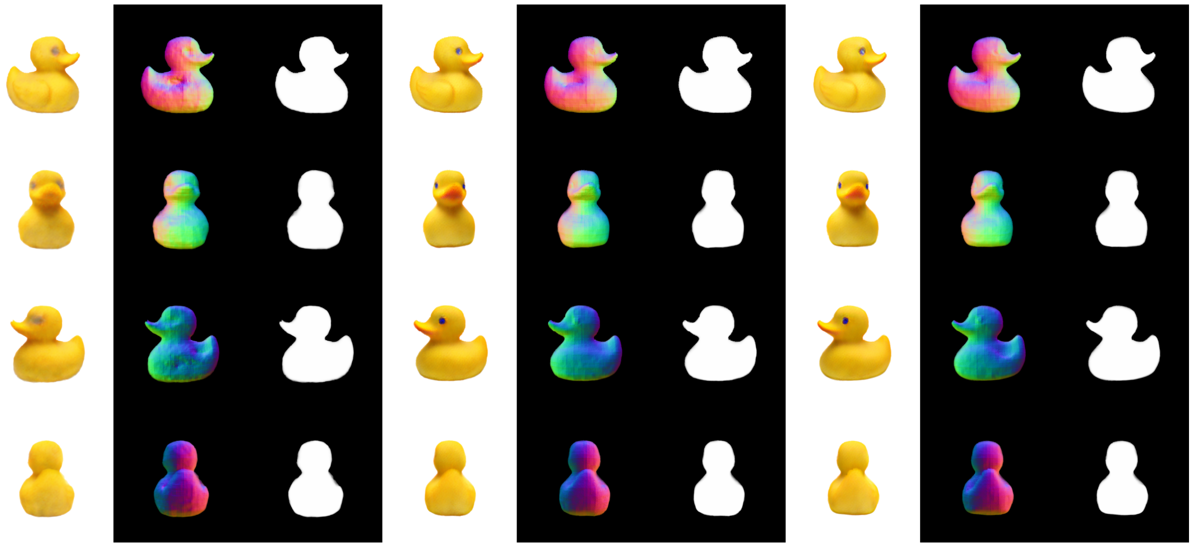


Figure A.12: Magic123 reconstructions of the duck after 10 min (left 3 columns), 30 min (middle 3 columns), and 60 min (right 3 columns) of synthesis.

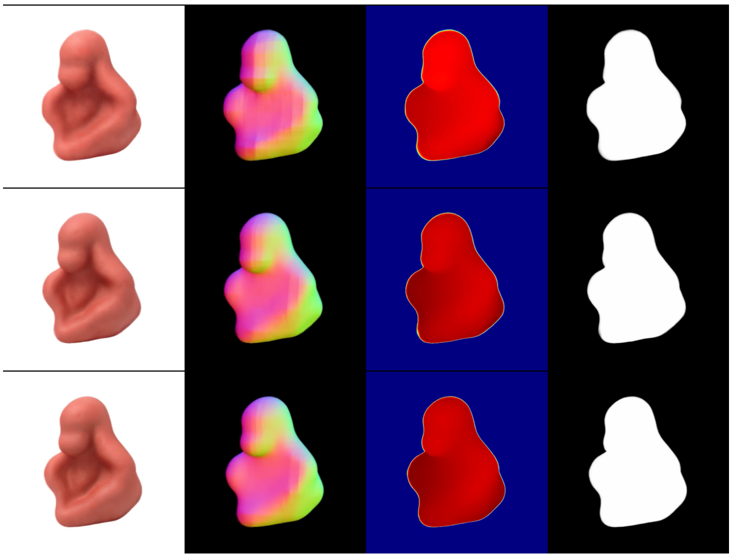


Figure A.13: Zero123 reconstructions of the gorilla prompt at the default settings, shown for 10 min (top), 30 min (middle), and 60 min (bottom) of generation.

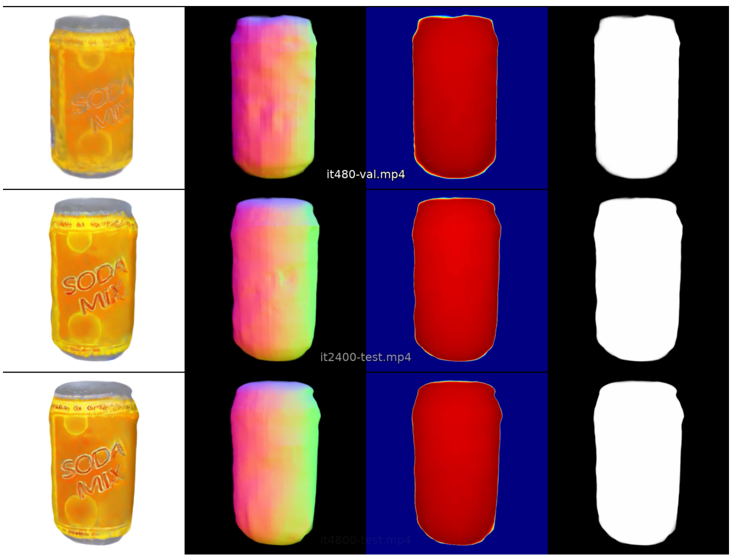


Figure A.14: Zero123 reconstructions of the soda can prompt at the default settings, shown for 10 min (top), 30 min (middle), and 60 min (bottom) of generation.

A.16 RQ2: Object-Specific Fidelity Metrics

Banana

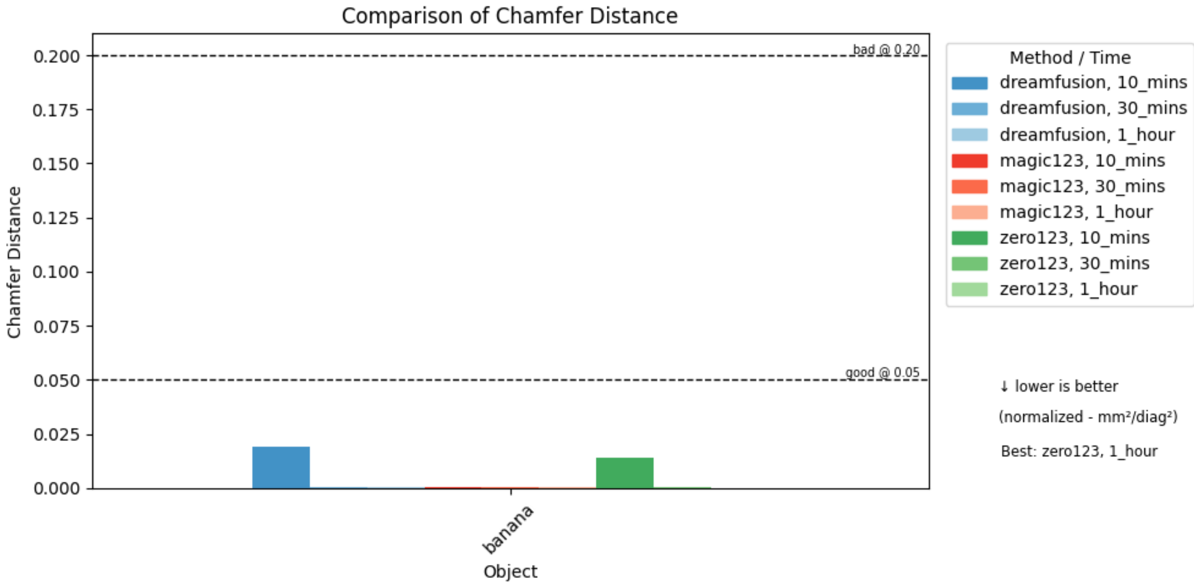


Figure A.15: Comparison of Chamfer Distance for all three methods and time budgets on the *banana* object. Lower is better (normalized $mm^2/diag^2$).

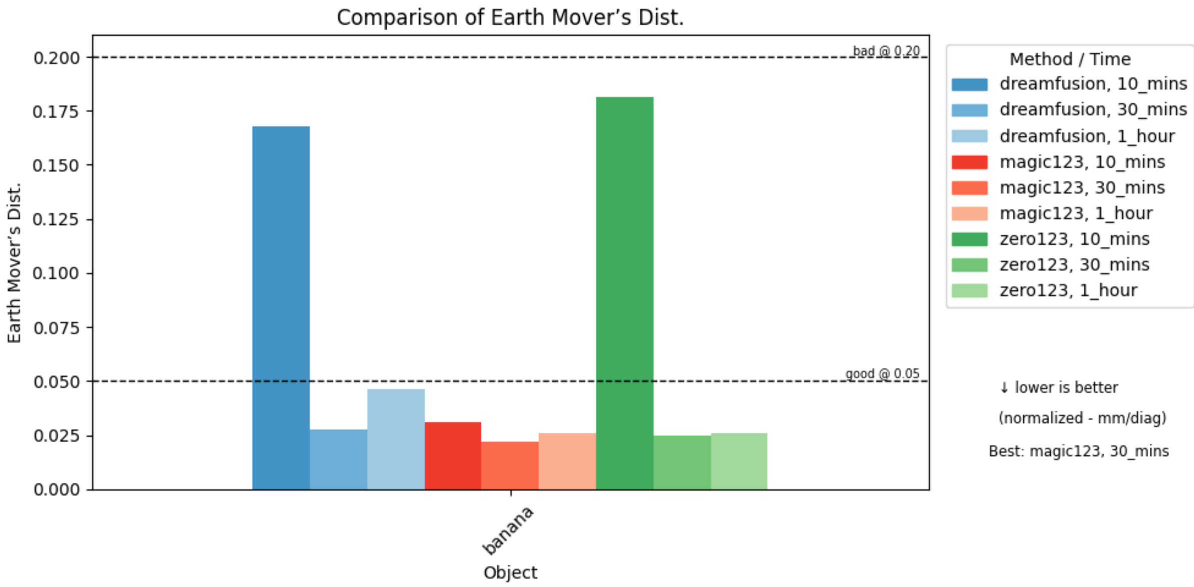


Figure A.16: Comparison of Earth Mover's Distance for all three methods on the *banana* object. Lower is better.

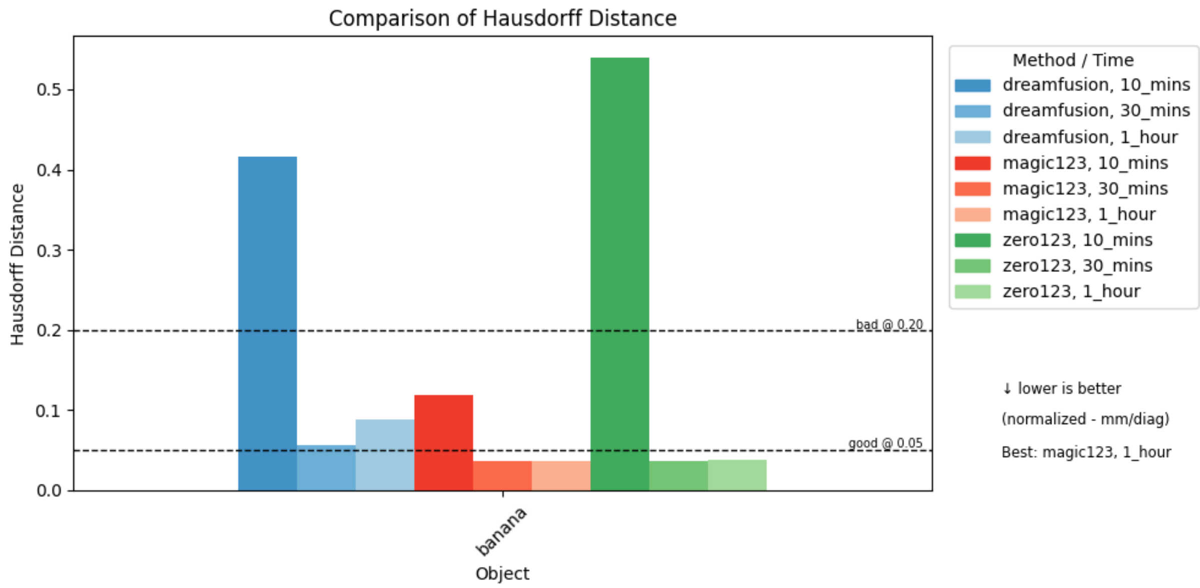


Figure A.17: Comparison of Hausdorff Distance for all three methods on the *banana* object. Lower is better.

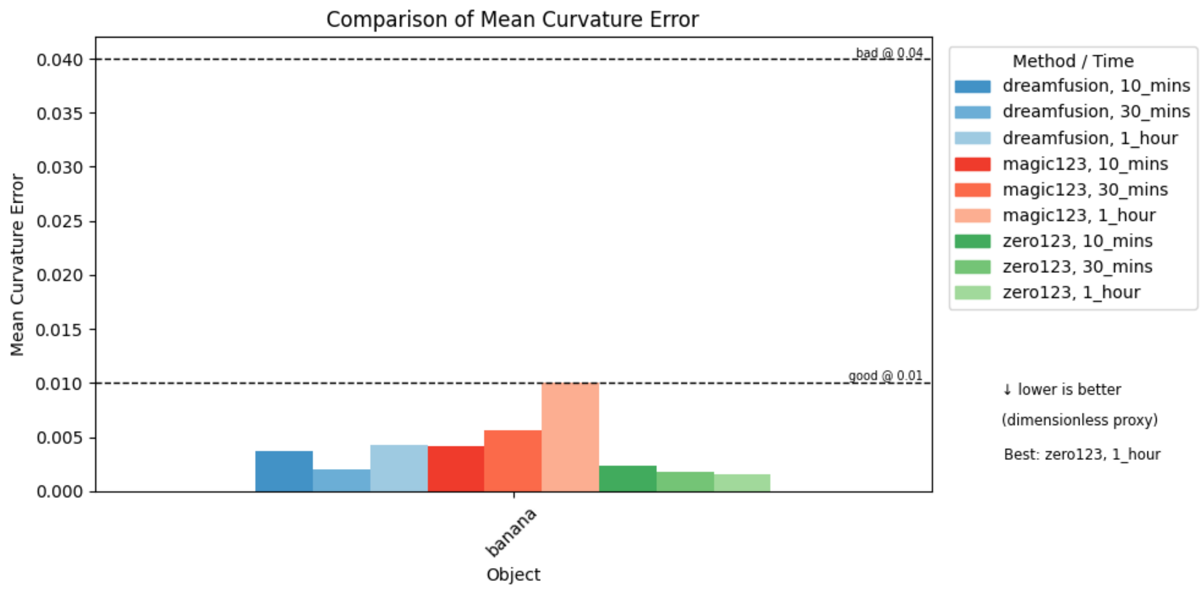


Figure A.18: Comparison of Mean Curvature Error for all three methods on the *banana* object. Lower is better.

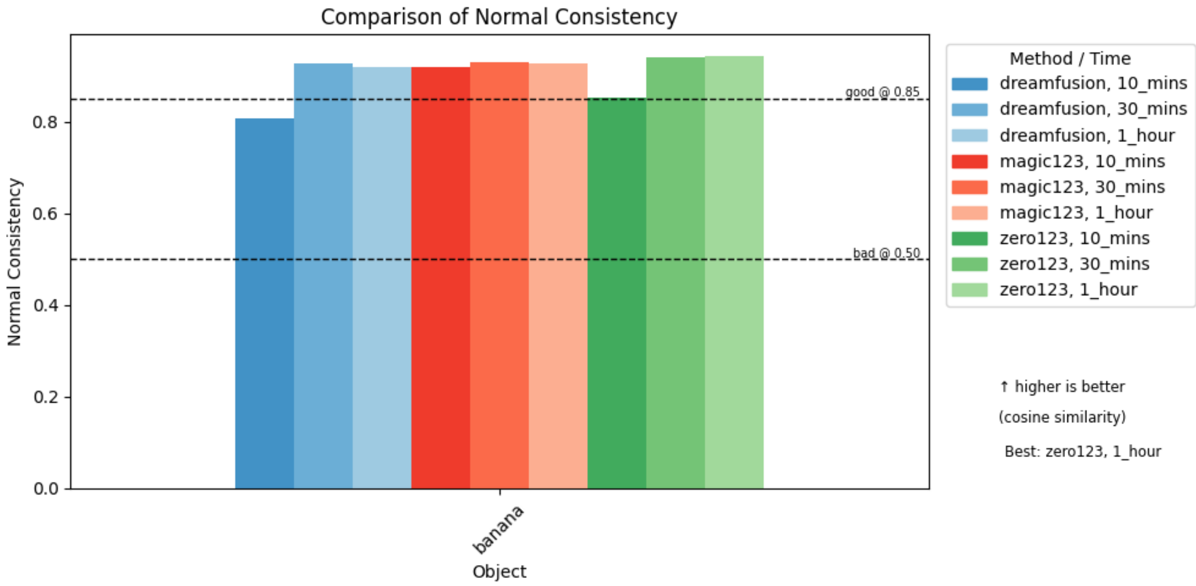


Figure A.19: Comparison of Normal Consistency for all three methods on the *banana* object. Higher is better (cosine similarity).

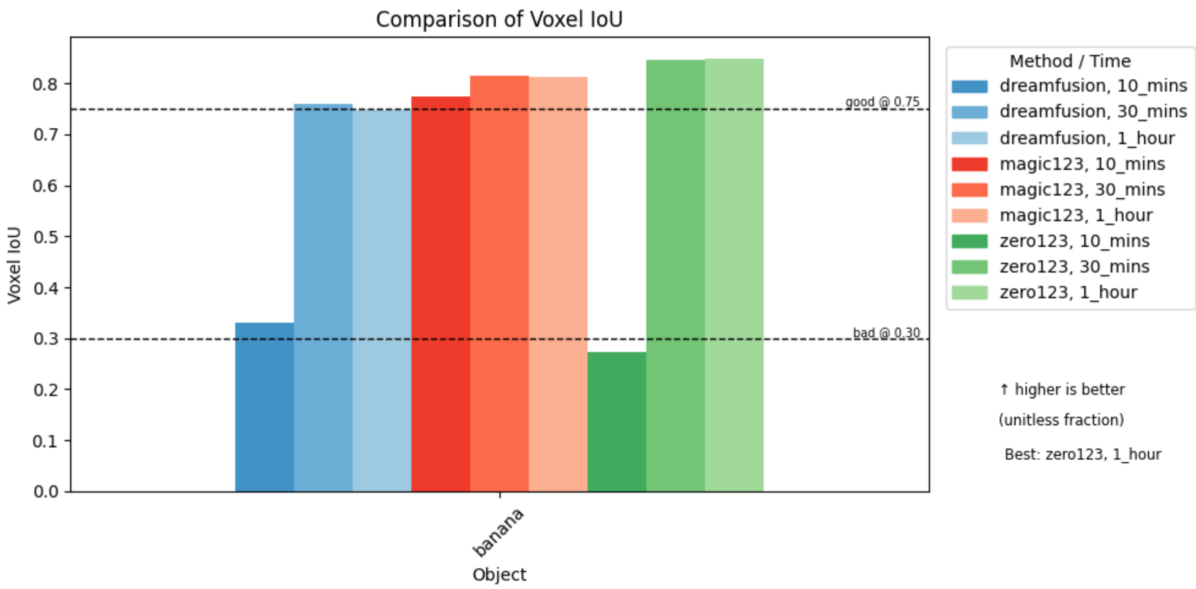


Figure A.20: Comparison of Voxel Intersection-over-Union for all three methods on the *banana* object. Higher is better.

Camera

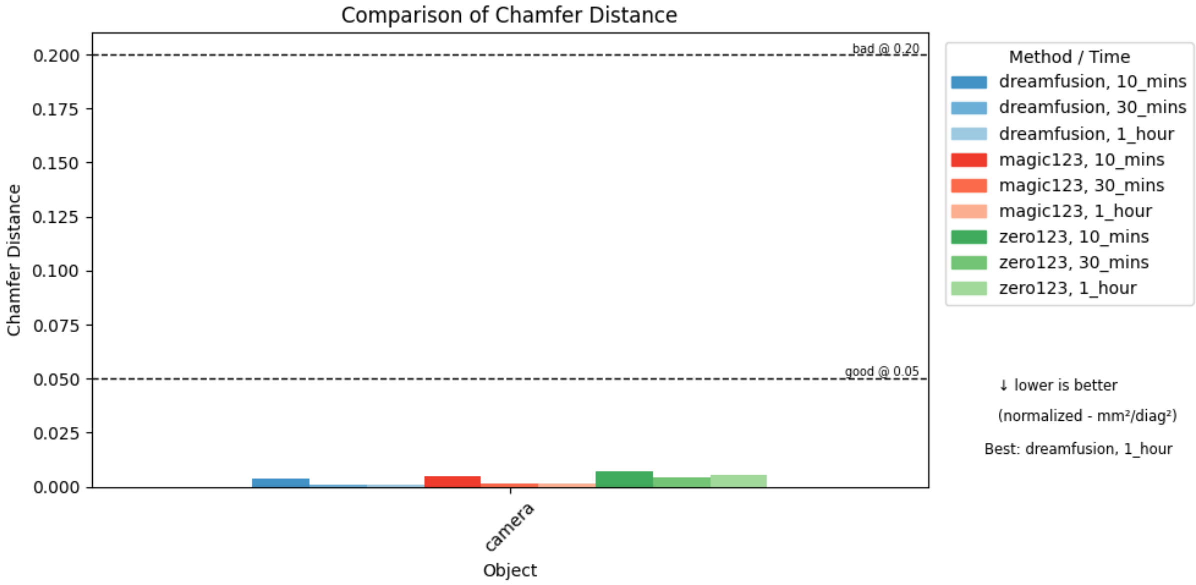


Figure A.21: Comparison of Chamfer Distance for all three methods and time budgets on the *camera* object. Lower is better.

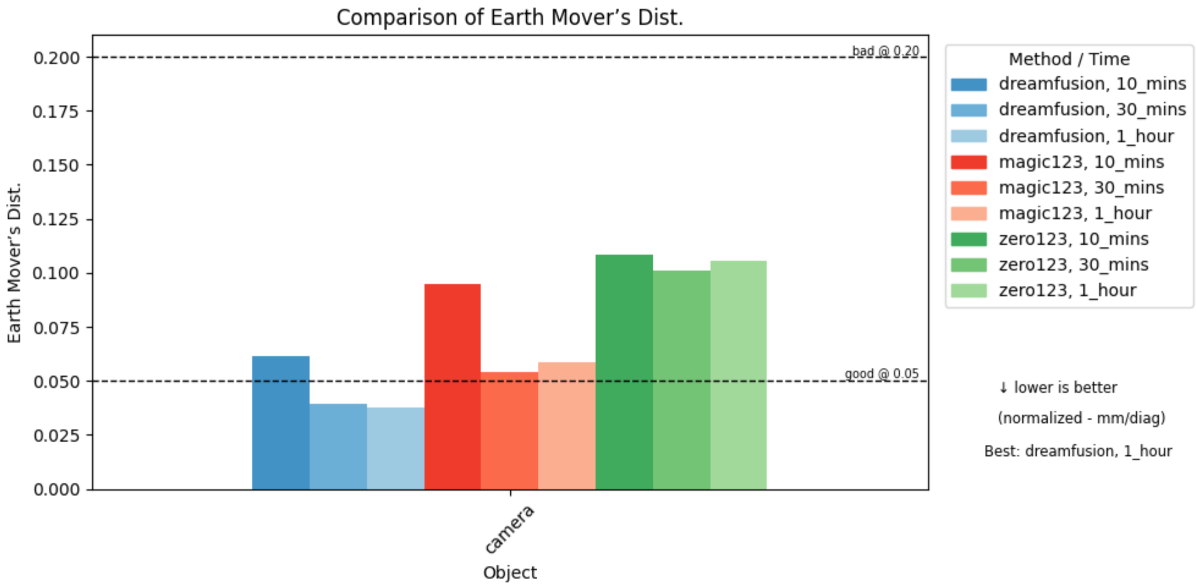


Figure A.22: Comparison of Earth Mover's Distance for all three methods on the *camera* object. Lower is better.

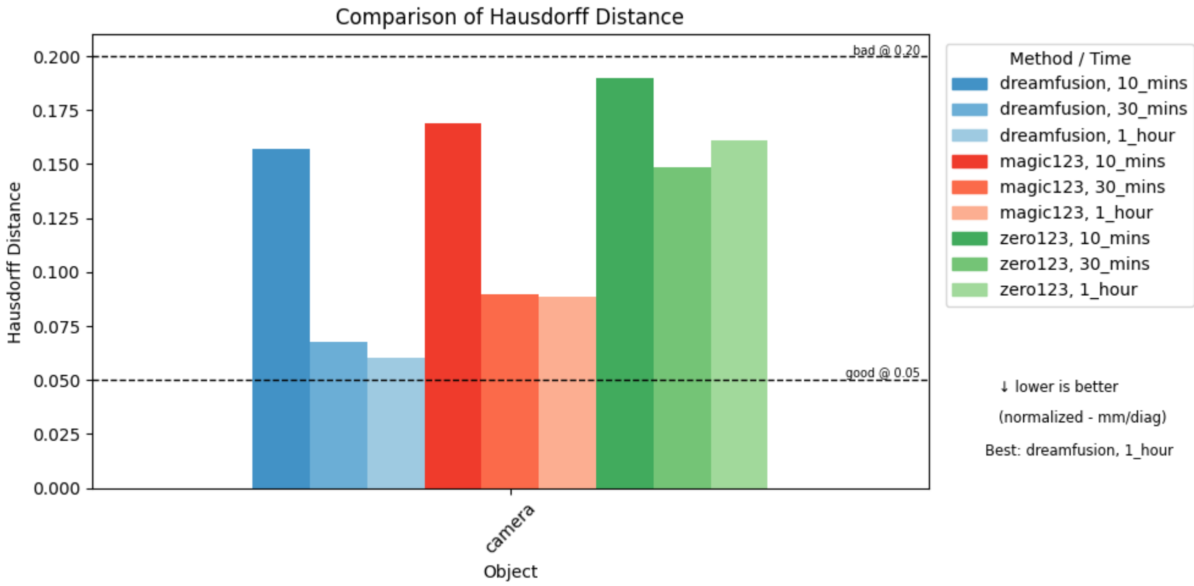


Figure A.23: Comparison of Hausdorff Distance for all three methods on the *camera* object. Lower is better.

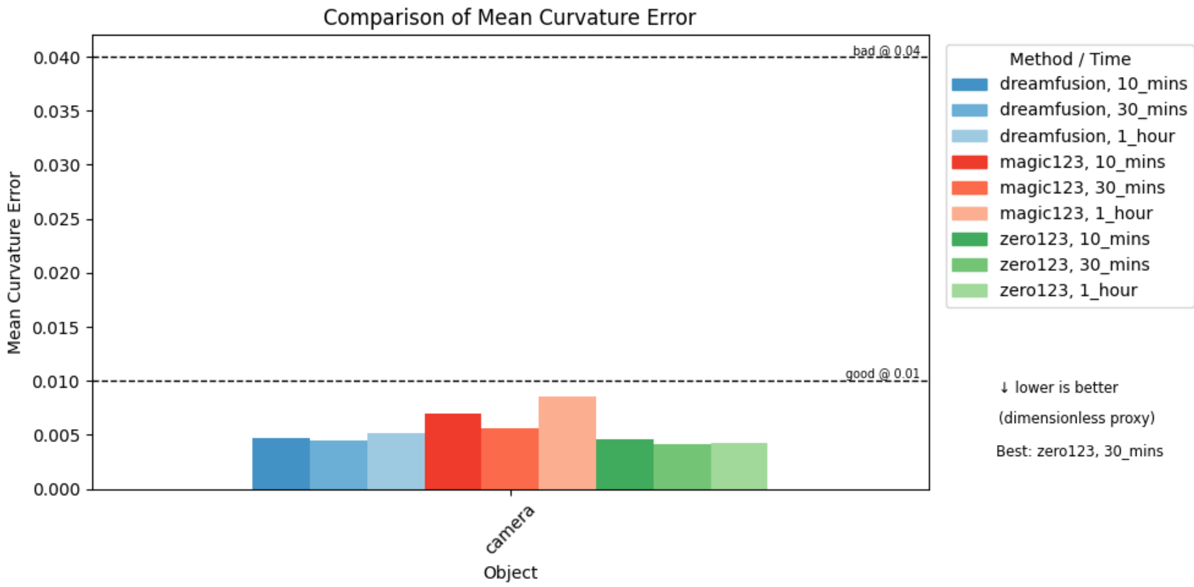


Figure A.24: Comparison of Mean Curvature Error for all three methods on the *camera* object. Lower is better.

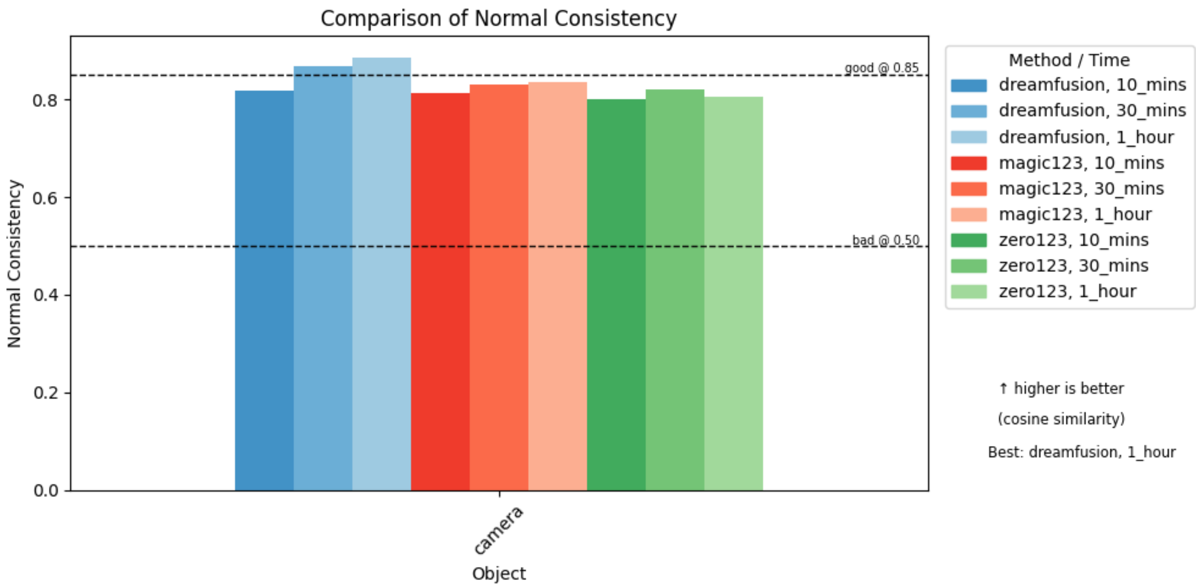


Figure A.25: Comparison of Normal Consistency for all three methods on the *camera* object. Higher is better (cosine similarity).

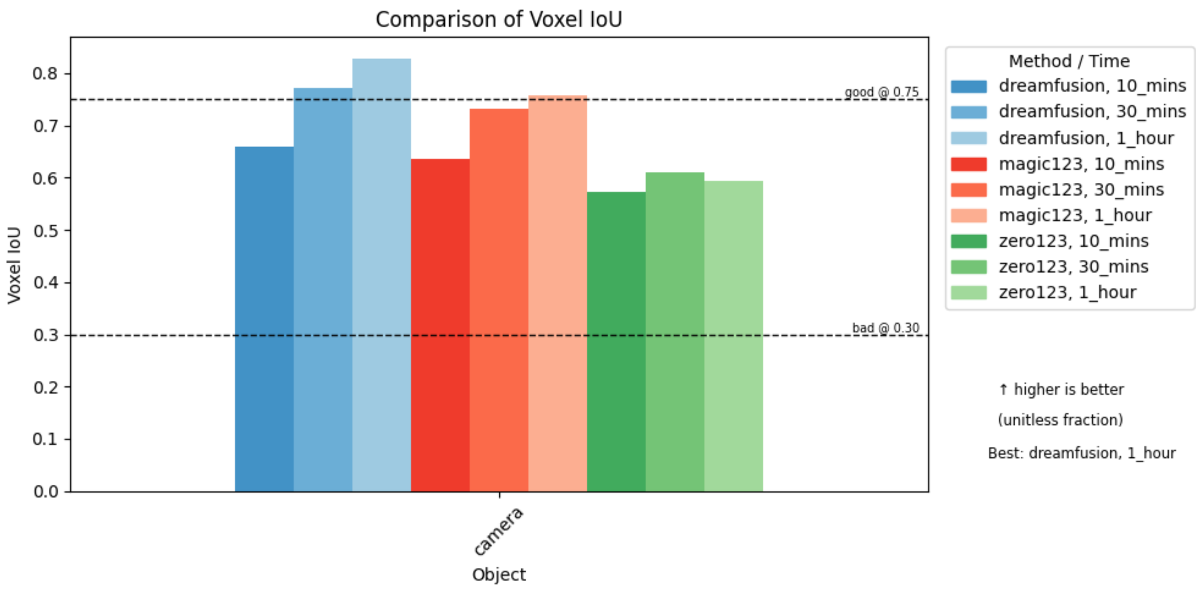


Figure A.26: Comparison of Voxel Intersection-over-Union for all three methods on the *camera* object. Higher is better.

A.17 RQ2: More detailed metrics per object

In this subsection of the appendix we report, for each object, which reconstruction method achieved the top score most often across our six fidelity metrics. By tallying wins per metric and selecting the model with the highest count, we identify the overall best GenAI reconstruction for each object.

Best method/model for each object

- **Banana:** Zero123 (4 votes) followed by Magic123 (2 votes)
- **Camera:** DreamFusion (5 votes) and Zero123 (1 vote)
- **Cat:** Zero123 (5 votes) and Magic123 (1 vote)
- **Duck:** DreamFusion (6 votes)
- **Gorilla:** Zero123 (4 votes) and Magic123 (2 votes)
- **Mouse:** Magic123 (4 votes) followed by both Zero123 (1 vote) and DreamFusion (1 vote)
- **Soda:** DreamFusion (6 votes)

Insights and interpretation The per-object "vote" analysis surfaces three key trends:

1. **Shape simplicity aids reconstruction.** Objects with smooth, symmetric geometry, most notably the Duck and Soda Can, were reconstructed flawlessly by DreamFusion (6/6 wins). This suggests that diffusion-based volumetric methods excel when the target has few sharp edges or fine surface details.
2. **Organic vs. mechanical complexity.** The "internet" objects (Banana, Soda Can, Mouse) generally proved easier: the Banana and Soda Can, both smooth, cylindrical forms, were best recovered by Zero123 and DreamFusion respectively. By contrast, the Camera (with its protruding lens, grip, and textured body) split votes (5/6 for DreamFusion, 1/6 for Zero123), indicating that intricate mechanical parts remain challenging for single-view generative pipelines.
3. **Model-specific strengths.**
   - **Zero123** dominated on highly articulated or organic shapes (Banana, Cat, Gorilla), winning 4–5 of 6 metrics. Its learned priors on everyday curved forms appear especially robust.
   - **DreamFusion** led on symmetric or rigid objects (Duck, Soda Can) and even the complex Camera, demonstrating its strength at enforcing global shape consistency.
   - **Magic123** shone on moderately detailed, partly curved objects (Mouse), suggesting its hybrid diffusion approach strikes a balance between smoothness and fine-feature reproduction.

Overall, no single method is universally superior. Instead, performance depends on object geometry: one might choose Zero123 for organic, free-form items, DreamFusion for symmetric or mechanical parts, and Magic123 for intermediate complexity.

A.18 RQ3: Outlier Analysis

Figure A.27 aggregates all eight per-frame outlier plots—four error metrics (rotation, translation, combined pose, ADD) over two groups of meshes. The top four rows show classical CAD perturbations:

1. original CAD
2. Gaussian noise
3. normal-aligned noise
4. speckle noise

and the bottom four rows show the sparse outlier spikes and the three generative-AI meshes:

1. sparse "spike" outliers
2. Zero123

3. Magic123
4. DreamFusion

Under the first four variants, red “extreme” points are vanishingly rare, confirming that small, uniform CAD perturbations have minimal impact on pose robustness. In contrast, Zero123 introduces occasional rotation and ADD spikes; Magic123 further increases both the frequency and magnitude of these failures; and DreamFusion exhibits the most pervasive outlier clusters—especially in rotation and ADD—underscoring FoundationPose’s sensitivity to coarse or noisy geometry.

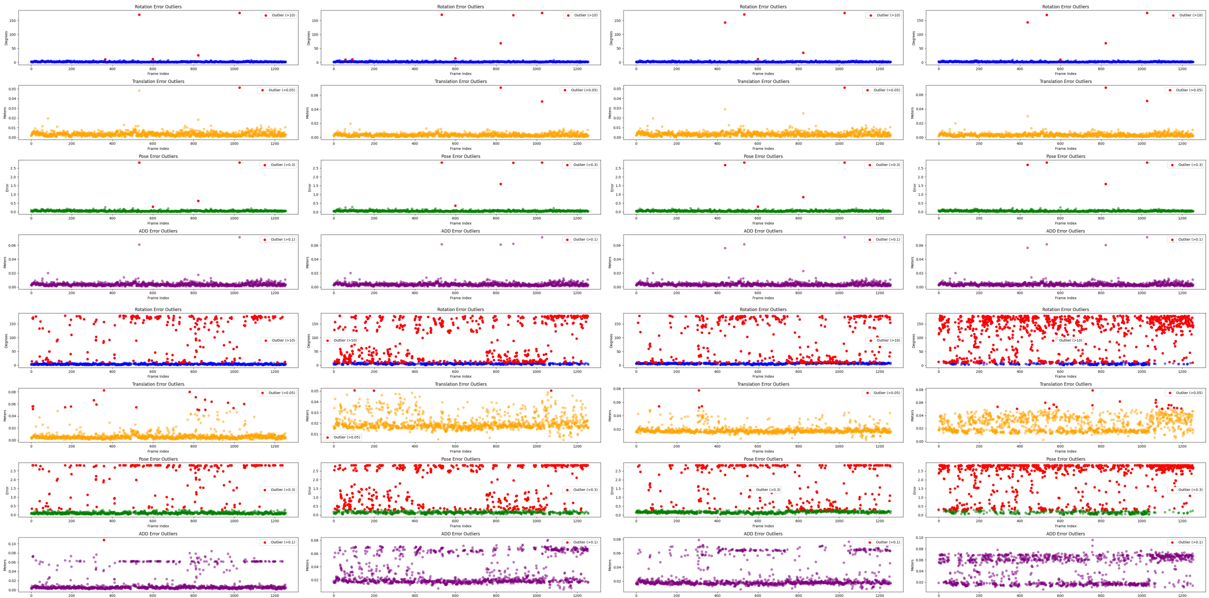


Figure A.27: Outlier distributions for four error metrics across eight mesh variants. First 4 rows & Columns (top left→ top right): (a) original CAD, (b) Gaussian noise, (c) normal-aligned noise. Last 4 rows & Columns (bottom left → bottom right): (d) speckle noise, (e) sparse “spike” outliers, (f) Zero123, (g) Magic123, (h) DreamFusion. Rows (top→bottom): 1. rotation error outliers, 2. translation error outliers, 3. combined pose error outliers, 4. ADD error outliers. In each subplot red markers denote values beyond the predefined outlier thresholds; blue/green/orange/purple points show in-threshold values.

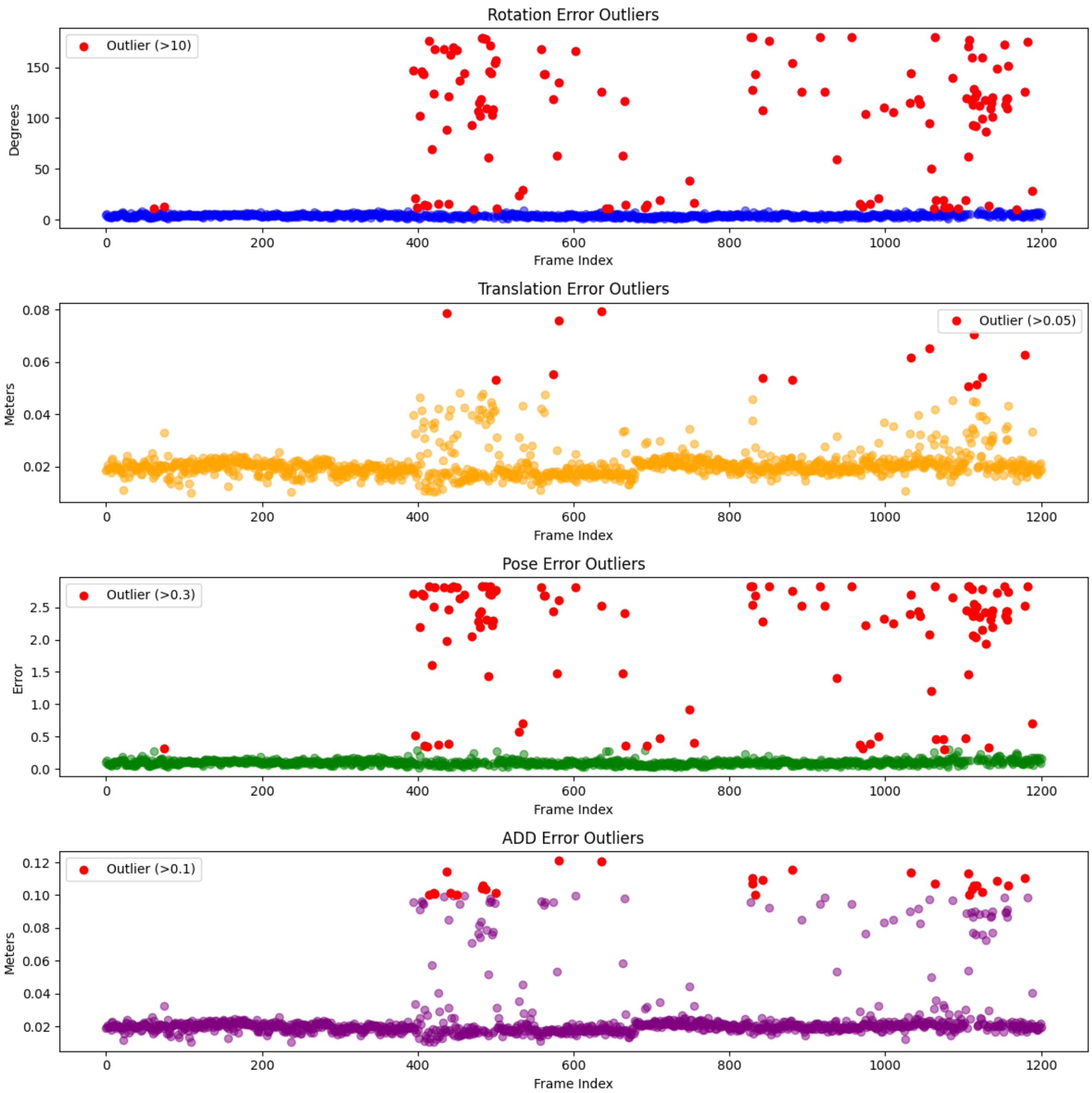


Figure A.28: Error outliers for the *camera* object (ID 4) under DreamFusion mesh. Note: frames 0–400 show near-zero errors, indicating that FoundationPose can succeed when the generative mesh and view align fortuitously, but after frame 400 large clusters of failures appear, reflecting poor overall mesh fidelity.

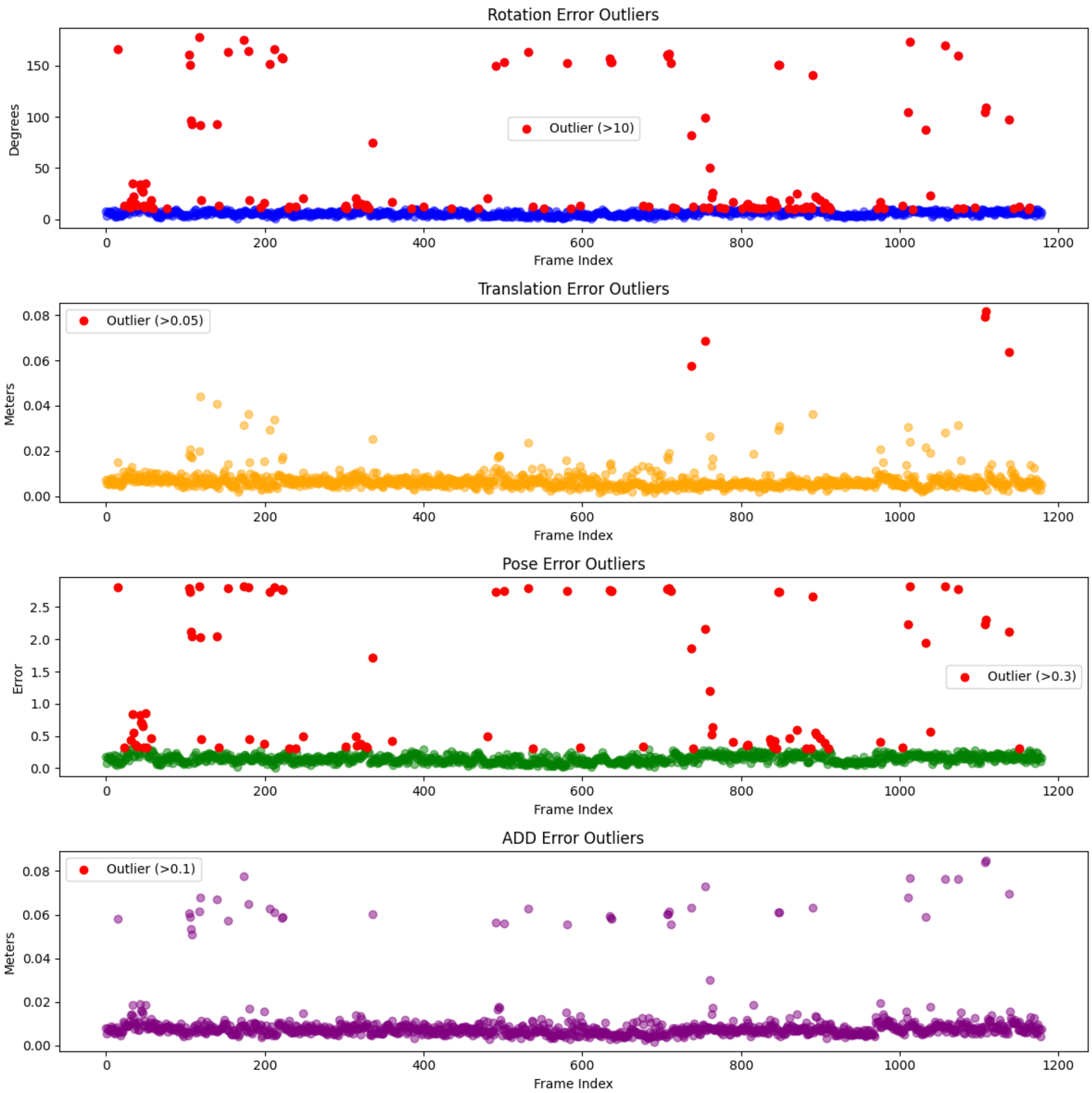


Figure A.29: Error outliers for the *cat* object (ID 6) under Zero123 mesh. Early frames exhibit low errors, showing that the rough Zero123 reconstruction occasionally matches the true view, but later frames spike, revealing that orientation-specific successes give way to widespread misalignment as viewpoint shifts.