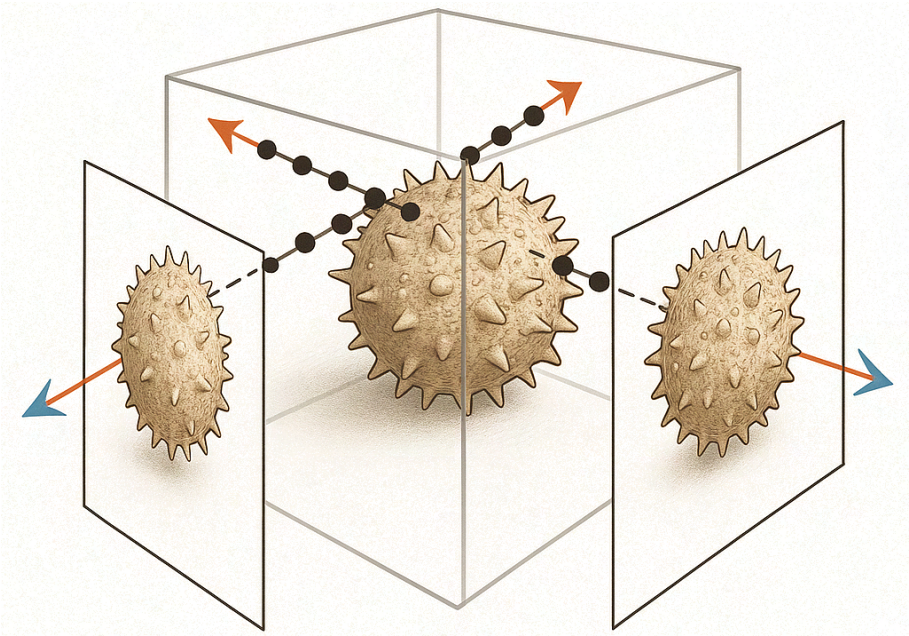
University of Applied Sciences and Arts Northwestern Switzerland
School of Computer Science

3D Reconstruction of Pollen Grains from Orthogonal Image Pairs

Bachelor's Thesis

Windisch, 14. August 2025



Authors	Nils Fahrni Etienne Roulet
Expert	Dr. Fabian Märki
Supervisors	Prof. Dr. Martin Melchior, FHNW Roman Studer, FHNW
Project No.	25FS_I4DS25

University of Applied Sciences Northwestern Switzerland, School of Computer Science

Abstract

Reconstructing 3D pollen grain shapes from extremely sparse visual data is a challenging problem, far more so than for many everyday objects, due to the fine spines, deep concavities, and vast morphological diversity of pollen. To date no learned 3D pollen reconstruction technique has emerged in the literature. The objective of this thesis is to investigate whether modern 3D reconstruction models can recover the full geometry of individual pollen grains using as few as two orthogonal input images.

We approach the problem as a controlled comparison across five model families that span the main 3D representations: (1) a geometric baseline (visual hull), (2) voxel CNNs (Pix2Vox), (3) mesh deformation (Pixel2Mesh++), (4) encoder-conditioned implicit fields (PixelNeRF), and (5) a large diffusion-based reconstructor used zero-shot (Hunyuan3D-2). We curate and augment the 3D Pollen Library (207 meshes), enforce watertightness and standardize outputs into a single evaluation pipeline to evaluate on common reconstruction metrics (Chamfer distance, F-Score@τ, IoU).

All contemporary approaches outperform our visual hull baseline. Under the two-view constraint, PixelNeRF delivers the best overall trade-off between accuracy and stability (e.g., CD $\approx$ 0.043, F-Score@5% $\approx$ 91%, IoU $\approx$ 83%). Hunyuan3D-2 in zero-shot two-view mode closely matches surface metrics (e.g., CD $\approx$ 0.043) but trails on volumetric overlap and requires substantially more compute. Pix2Vox recovers plausible coarse volume but blurs fine ornamentation at 32^3. Pixel2Mesh++ achieves competitive surface proximity yet produces non-watertight meshes in many cases; using a mean-shape template improves IoU but not surface error. Ablations show the largest gain from one to two views, with diminishing returns thereafter.

Keywords: 3D reconstruction, pollen grains, computer vision, deep learning, neural rendering, volumetric modeling, shape estimation, few-shot learning, implicit representation, NeRF-based reconstruction, volumetric rendering

The code for this thesis can be found at `https://github.com/3d-pollen-reconstruction/sequoia`

Acknowledgements

The completion of this thesis and our success at FHNW were made possible by the support and encouragement we received from those around us.

We are deeply grateful to Prof. Dr. Martin Melchior and Roman Studer, our supervisors, for their steady guidance and thoughtful questions. Our weekly meetings were an anchor throughout the project, helping us reflect on our ideas, refine our approach, and keep our thesis focused and rigorous. We also gratefully acknowledge the i4DS institute for providing the computational resources that enabled our experiments.

We hope that the experiments and lessons gathered here provide good starting points and useful context for future efforts in this new domain of 3D pollen grain reconstruction at i4DS, and that our research has a positive impact on automated pollen monitoring, allergy forecasting, and related environmental health applications.

We extend our appreciation to all the professors who supported us behind the scenes throughout our Bachelor's program, culminating in this thesis. We are equally thankful to our friends and peers for the many discussions that clarified our thinking and for their encouragement during the more challenging stages of this thesis.

Finally, we owe heartfelt thanks to our families for their patience, constant support and belief in us. Their encouragement has sustained us throughout our studies.

Contents

List of Figures

List of Tables

1 Introduction

A fine dust to the naked eye, pollen grains are intricate in microscopic form and fundamentally important to life on Earth. Pollen grains carry male reproductive cells from the anthers of flowers, making them indispensable for plant reproduction and vital to both natural ecosystems and our food systems [1]. Beyond their role in plant reproduction, pollen grains provide durable records that inform reconstructions of plant evolution and past environmental change. At the same time, pollen can have adverse health effects: an estimated one-third of the global population experiences pollen-induced respiratory allergy (hay fever) [1], so seasonal increases in airborne pollen constitute a recurring public health burden.

The dual role of pollen, as essential to plant reproduction and a common trigger of allergic disease, motivates precise monitoring. Reducing the public-health burden of allergic hay fever requires accurate, species-specific information on airborne pollen. A high overall "pollen count" conveys limited value; distinguishing whether concentrations are dominated by weakly allergenic pine versus highly allergenic ragweed is clinically and epidemiologically more informative.

Traditionally, monitoring relies on manual microscopy: airborne particles collected on substrates are identified and counted by trained analysts. Although this approach is well established, it is slow, labor-intensive, and subject to inter-observer variability. Recent automated monitoring systems aim to address these limitations. One such approach uses digital holography to record two-dimensional interference patterns of individual particles in flight.

A raw hologram is not a direct image of the grain but an interference pattern that encodes the complex optical field and, implicitly, the three-dimensional structure. Recovering a usable representation requires computational reconstruction. The quality of this reconstruction forms the bridge between data acquisition and subsequent classification: diagnostic morphological features (e.g., pores, furrows, and exine texture) must be resolved with sufficient fidelity for machine-learning models to discriminate among morphologically similar taxa.

This thesis concentrates on computational reconstruction. We evaluate methods that transform raw two-dimensional holograms into accurate three-dimensional representations of pollen using computer vision and deep learning. By improving reconstruction fidelity, we aim to enhance the performance of downstream tasks such as species-level classification and, ultimately, to increase the reliability of automated pollen monitoring.

1.1 Motivation

Technologies like digital holography, introduced to palynology via the Swisens Poleno by Sauvageat, Zeder, Auderset, *et al.* [2], capture two orthogonal images from a hologram. While these systems provide the raw imaging data, the task of generating accurate and detailed 3D models from a minimal number of 2D projections remains a non-trivial challenge. Moreover, downstream tasks such as pollen classification could greatly benefit from volumetric representations generated by 3D reconstruction models.

In this thesis, we are motivated by this challenge and seek to validate methods for 3D pollen reconstruction under severe data constraints. We explore the feasibility of generating 3D models from only two orthogonal 2D images of a pollen grain. By testing and comparing different reconstruction approaches under these stringent conditions, we aim for this thesis to serve as a foundational study assessing whether these models can produce meaningful 3D representations.

1.2 Goals

The central aim of this thesis is to determine whether any existing 3D reconstruction approach can be adapted to recover the complete 3D shape of individual pollen grains from as few as two mutually orthogonal images. To address this aim, the work will:

1. Identify suitable model families drawn from current single- and few-view 3D reconstruction literature, selecting only those that can plausibly recover the geometry/volume of pollen grains.
2. Systematically adapt and train each selected model on paired orthogonal images of pollen, keeping training conditions as uniform as possible to enable fair comparison.
3. Compare the adapted models in controlled experiments, using common geometry-focused metrics (e.g., Chamfer distance, IoU) to quantify reconstruction fidelity.

An important remark is that the goal is to reconstruct the general geometry and volume of pollen grains. Fine textures are less important in this work because this thesis mainly serves as a preliminary comparative study. The main goal remains, namely to show whether 3D reconstruction of pollen grains is possible with just two orthogonal images as an input prior.

1.3 Research Questions

We present three primary research questions and one additional question designed to assess our methods beyond a synthetic dataset by applying them to real-world holographic imagery.

1. What 3D reconstruction technique is best suited when restricted to two orthogonal images as an input?
2. Do explicit priors of pollen shapes improve the quality of 3D reconstructions from 2D images?
3. To what extent is the reconstruction quality restricted by having only two orthogonal images?
4. *(Bonus)*: Can a 3D reconstruction approach trained on synthetic models be adapted to handle real-world holographic images of pollen grains?

1.4 Contributions

This thesis brings 3D computer vision into palynology by showing, for the first time, that few-view reconstruction models can be adapted to recover the full geometry of pollen grains. However, virtually no dedicated 3D datasets exist for pollen 3D reconstruction. We curate a sparse collection of 3D pollen grain meshes, tailor it to the constraints of the Swisens Poleno device [2], train 3D reconstruction models, and benchmark them. The resulting comparison exposes which model family handles the data scarcity and viewpoint limitations best. All code, image pairs, trained weights, and evaluation scripts are released publicly to encourage further exploration of 3D learning on pollen grains.

1.5 Document Structure

We structure the thesis top–down from the learning objective to the data it requires. We first establish what has to be predicted and how success is measured, and only then derive the dataset interfaces and preprocessing from those requirements. This ordering is intentional: the models we study (voxel CNNs, mesh deformation, encoder-conditioned NeRF, and a large diffusion prior) expect different input conventions and produce heterogeneous outputs that we standardize to representations under a unified evaluation pipeline. We therefore define *Models and Evaluation Procedures* before *Data and Preprocessing*.

1. **Background & Related Work (Section 2)**: Maps the space of 3D representations and pollen imaging. We need this background to motivate the specific reconstruction families we evaluate in this thesis.
2. **Models and Evaluation Procedures (Section 3)**: Justifies the shortlist under our hardware constraints and two-view inference regime. It details each method's inputs/outputs and fixes a common evaluation protocol (ICP alignment, Chamfer, F-Score, IoU). By committing here to a unified mesh standard and metrics, we precisely determine what the data pipeline must deliver later.
3. **Data and Preprocessing (Section 4)**: Starts from the 3D Pollen Library and builds a base dataset, then derives *model-specific* preprocessing dictated by Section 3 (e.g., voxelization and multi-view synthesis for Pix2Vox, mesh simplification and canonicalization for Pixel2Mesh++, calibrated view/pose sampling for PixelNeRF). We ensure every method consumes comparable inputs and yields meshes compatible with our evaluation.
4. **Experiments and Results (Section 5)**: Organizes experiments to answer the research questions: (RQ1) compare model families under two orthogonal views; (RQ2) test explicit pollen priors; (RQ3) analyze sensitivity to number of views; (RQ4) assess zero-shot transfer to holographic images using a large reconstruction model. We also add more general experiments to explore the benefits of transfer learning as well as to justify data augmentation.
5. **Discussion & Limitations (Section 6)**: Interprets the findings with respect to RQs and domain constraints, and outlines limitations.
6. **Future Work (Section 7)**: Proposes concrete next steps to improve current pipelines, explore large reconstruction models, and expand datasets (including holography).
7. **Conclusion (Section 8)**: Summarizes evidence-based answers to the RQs and implications for automated pollen monitoring.
8. **References and Appendix**: Bibliography and supplementary material (extended qualitative/quantitative results, hyperparameter sweeps where applicable, and further implementation details).

2 Background & Related Work

In this section we first map out the different 3D representations that objects can be rendered in. With these, we establish an understanding of how deep learning models can learn from such geometries and ultimately reconstruct objects. In a second step, we give an overview of current pollen imaging methods to learn how devices like the Swisens Poleno measurement devices capture pollen. After building this background, we look at the landscape of state-of-the-art approaches for 3D reconstruction.

2.1 3D Object Representations

Before we dive into pollen imaging techniques and 3D reconstruction techniques, we first consider how objects can be represented in a way that is learnable by deep learning models. We discuss this view detached from our selected pollen dataset, which we will tackle later in Section 4, before presenting our experiments.

In our case, there are four relevant types of representations in which we can portray 3D objects:

- **Meshes**. A mesh represents a 3D surface using vertices (points in 3D space) connected by edges to form faces, typically triangles. The collection of faces approximates the shape of an object's surface. Meshes are relatively efficient for rendering and can capture fine geometric detail while keeping storage relatively compact. However, they only encode surfaces, not the interior of objects, and they require connectivity information between vertices.
- **Point Clouds**. A point cloud is a set of 3D points sampled from the surface of an object or scene. Each point can store additional attributes such as the surface normal. Point clouds are simple and flexible but lack explicit surface connectivity, which makes it harder to render smooth surfaces or compute object boundaries without additional processing. They are closely related to meshes as meshes can essentially be represented as point clouds when we preserve each triangle's three points.
- **Voxels**. Voxels are the 3D analogue of pixels: small cubic cells arranged in a 3D grid. Each voxel can store binary occupancy (filled or empty). Voxel grids provide a uniform and straightforward representation of volume, including both surface and interior, but their memory cost grows cubically with resolution. This makes high-detail reconstructions computationally quite expensive.
- **Neural Radiance Fields**. Unlike the previous three representations, a Neural Radiance Field is a continuous, implicit representation of a 3D scene, learned by fitting a neural network to predict the color and density of any point in space, given a viewing direction. This means, instead of directly obtaining an object, a Neural Radiance Field is a function that describes the object. By integrating the predictions of this function or neural network along camera rays, images from arbitrary viewpoints can be synthesized. NeRFs capture fine details and view-dependent effects without storing explicit geometry, but rendering and training are computationally intensive, and extracting an explicit mesh or point cloud requires additional processing.

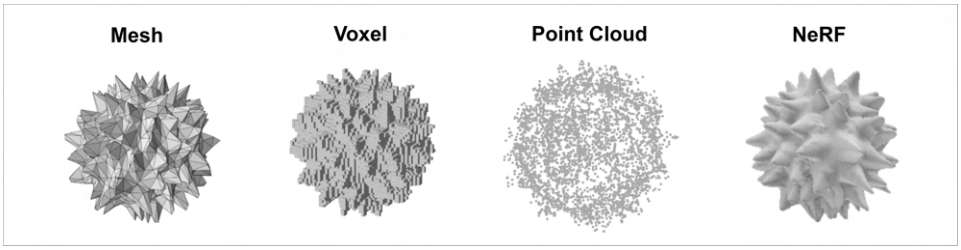


Figure 2.1: Four common representations showing a pollen grain: (1) Mesh, composed of connected triangles, (2) Voxel grid, built from small cubic elements, (3) Point cloud, consisting of sampled surface points, and (4) Neural Radiance Field (NeRF), producing a smooth, realistic rendering by simulating light rays through an implicit function.

We understand that in the landscape of 3D reconstruction, there are more than these four types of 3D representations. We selected these as they are the most common in literature. Additionally, these representations are the ones used in our selected models, which we will be defining later in Section 3.1.

2.2 Pollen Imaging and Classification

Before diving deeper into the landscape of machine-learning-based reconstruction methods, we give an overview of how pollen grains are imaged in the real world.

Pollen grains are microscopic particles. To understand their morphologies numerous imaging techniques have been developed and are worth noting, so we can draw a connection to the constraints of the images used to reconstruct pollen grains later.

Understanding pollen morphology is interesting for fields ranging from allergy management to climatology [3]. Traditionally, such imaging methodologies for pollen analysis can be broadly categorized into three meta categories: optical microscopy, electron microscopy, and advanced computational methods.

Optical microscopy uses visible light to examine pollen grains. Optical microscopy techniques like confocal microscopy can capture the overall 3D shape and internal structures but struggle to clearly resolve fine surface textures. On the other hand, methods such as Differential Interference Contrast (DIC) microscopy work well at highlighting surface details but provide only limited 3D structural information. Super-resolution microscopy methods, such as Structured Illumination Microscopy (SR-SIM), bridge this gap somewhat, providing finer detail approaching that of electron microscopy, yet remain complex and resource-intensive [4].

Electron microscopy (EM) produces very high-resolution images. Scanning Electron Microscopy (SEM) clearly shows the surface features of pollen but requires coating the samples with metals like gold, making it destructive and unsuitable for examining internal structures [5], [6]. Transmission Electron Microscopy (TEM) can capture internal details at a high resolution but only works on extremely thin sections of pollen. This means EM methods offer no complete 3D reconstruction capability and make the sample preparation cumbersome and destructive.

Computational imaging techniques incorporate machine learning to enhance or automate pollen analysis. Imaging Flow Cytometry (IFC) [7], for example, rapidly produces thousands of pollen grain images per second and can classify them automatically using deep learning algorithms. However, IFC primarily produces two-dimensional images and uses fluorescence as an additional trait for identification. This limits its application since a clear 3D shape is quite expensive to recover from these images [7]. Recent advancements like digital holography offer

real-time, non-destructive imaging but still face challenges in providing complete and accurate 3D structural details.

Reconstructing pollen grains in their full 3D morphology with deep learning remains an unexplored task. Research at the intersection of palynology and machine learning has shown that generative models like StyleGAN can be used to create synthetic pollen images Mills, Zervas, and Grant-Jacob [8]. By manipulating the model's latent space novel images with controlled features such as size, ornamentation, and shape, can be generated. This directly allows for the simulation of morphological variations and transformations between different taxa. The research of Mills, Zervas, and Grant-Jacob [8] shows that the structure of pollen grains can indeed be learned by conventional deep learning models, however, it has only been explored for 2D image representations and not 3D representations.

Due to the limitations of the aforementioned imaging techniques, acquiring accurate 3D representations of pollen grains remains a complex and expensive task. Current methods either damage the samples, are limited to two-dimensional images, or require significant manual preparation and sophisticated equipment.

2.3 Methods used in 3D Object Reconstruction

Now that we discussed the difficulty of obtaining 3D pollen grains using common imaging techniques, we also need to look at our task from the perspective of 3D reconstruction models.

Reconstructing 3D shape from 2D images is a longstanding challenge in computer vision. Traditional multi-view geometry techniques (e.g. structure-from-motion and multi-view stereo) can recover detailed 3D models given many images with overlapping views. However, these classical methods struggle when only a few images are available, especially if the views are widely separated [9].

In recent years, deep learning has transformed 3D reconstruction by enabling priors and recognition capabilities. Rather than purely relying on geometric triangulation, neural networks can learn the space of likely 3D shapes from training data and thereby "hallucinate" plausible geometry even in occluded or unseen regions [9]. Widely-used benchmark datasets like ShapeNet [10] (a repository of synthetic 3D models) have accelerated this development, providing thousands of exemplars for training and evaluation. For example, Choy, Xu, Gwak, *et al.* [11] introduced 3D-R2N2, an early deep model that uses a recurrent neural network to integrate multiple views of an object and output a voxelized 3D volume. This model and others from the same period demonstrated that neural networks can predict reasonable 3D shapes from one or more images, given sufficient training on similar objects. They intrinsically learn shape priors of the dataset which then gets formed into a plausible shape based on the input image. However, the quality of these reconstructions was limited by the chosen 3D representation (e.g. low-resolution voxels) and by the difficulty of the task, especially in the extreme case of only one or two input images.

2.4 Learned Representations for 3D Shape

A key difference among learning-based approaches is the choice of 3D representation. Early works, such as 3D-R2N2 [11], often predicted volumetric grids (voxels), which have the advantage of a regular structure but suffer from coarse resolution and high memory usage. For instance, the 3D-R2N2 model produced 32^3 voxel grids and could merge information from multiple views, but its outputs often lost thin structures and surface details. Kar, Häne, and Malik [12] similarly used volumetric representations (in a method sometimes referred to as Learnt Stereo Machines (LSM)) and faced the same issue of missing fine details. As a result, researchers explored alternate representations: point clouds (sets of 3D points of any arbitrary coordinate) which

are memory-efficient but unstructured, meshes which explicitly model surfaces can represent continuous geometry and rely on such point clouds as points where the edges of these surfaces meet. Each representation has trade-offs. Point clouds can capture detail but may require post-processing to produce a solid surface, whereas meshes directly provide surface connectivity but are harder to predict directly with neural networks.

2.5 Sparse-View Reconstruction and Neural Rendering

Even as mesh- and voxel-based models improved, a parallel revolution was underway in neural rendering and implicit representations. The seminal Neural Radiance Fields (NeRF) approach by Mildenhall, Srinivasan, Tancik, *et al.* [13] showed that a multilayer perceptron can represent a 3D scene as a continuous field of color and density. This enables highly realistic novel view synthesis via volumetric rendering. However, classic NeRF requires dozens of images per scene and a lengthy per-scene optimization to fit the radiance field. This is impractical in sparse-view scenarios. Subsequent research attacked this limitation from two angles: reducing the number of required views, and avoiding per-scene training by learning priors over many scenes.

2.6 Diffusion Models and Large-Scale 3D Generation

Most recently, the field of 3D reconstruction has seen a surge of interest in large models and diffusion-based approaches for 3D reconstruction and generation. These approaches leverage massive training corpora and powerful architectures (often transformers in combination with diffusion networks) to produce remarkable 3D outputs from minimal input. One notable example is Tencent's Hunyuan3D-2 system [14]. Hunyuan3D-2 is a two-stage pipeline: first a shape generator produces a detailed mesh that aligns with the input (which could be one or more images, or even a text prompt), then a texture generator produces high-resolution textures for that mesh. The shape generator is built on a diffusion transformer, essentially a large-scale denoising diffusion model trained to generate an object that is seen on a single conditioning image. The result is a highly detailed 3D asset that not only fits the provided view(s) but is also textured, enabling realistic rendering. Hunyuan3D-2 has been reported to outperform prior models in geometry detail, alignment to inputs, and texture quality. It represents the state-of-the-art in combining generative AI with 3D reconstruction.

Several open-source efforts in 2024 have followed a similar trend, namely, to generate 3D objects from one or few images in a feed-forward pass (as opposed to optimizing a model per instance). TripoSR [15] and Stable Fast 3D (SF3D) [16] are two such models, developed in part by collaborations with Stability AI. These methods leverage transformer-based architectures (sometimes referred to as LRM: Large Reconstruction Model architectures) to achieve both speed and quality. For example, TripoSR uses a transformer to directly regress a 3D mesh from a single image in under 0.5 seconds [15]. By training on vast datasets with improved data processing and model design, TripoSR achieved superior performance compared to previous open-source models.

Similarly, SF3D explicitly tackles the problem of textured mesh reconstruction from a single image, and introduces a fast texture unwrapping and texture generation mechanism. Instead of predicting vertex colors (which many older networks did, resulting in unwrapped textures tied to geometry), SF3D learns to generate a texture map and even predicts material parameters and normal maps to enhance realism. A post-processing de-lighting step ensures the texture is not biased by lighting in the input image, which makes the output more adaptable to new illumination. SF3D's emphasis on texture as well as geometry is important for applications that require not just a shape but a fully ready 3D asset. In evaluations, SF3D demonstrated superior performance over existing techniques on both geometry and appearance fidelity. What's

remarkable is the speed: these models produce results in seconds or less, making them practical for interactive use or large-scale processing.

Another cutting-edge approach is InstantMesh [17], which combines the power of diffusion models with a dedicated reconstruction network. InstantMesh uses an off-the-shelf multi-view image diffusion model to first generate additional views of the object (given a single input image). In essence, it hallucinates how the object would look from other angles. This is a task made feasible by diffusion models trained on billions of images (one such model is Zero-1-to-3, which is a zero-shot view synthesis model based on Stable Diffusion [18]). These generated views, along with the original image, are then fed into a sparse-view 3D reconstruction network (built on the aforementioned LRM architecture) which produces a final mesh. By "synergizing the strengths" of generative view synthesis and learned reconstruction, InstantMesh attains excellent results in both diversity and quality of 3D assets. Impressively, it can create a 3D mesh in around 10 seconds, significantly faster than prior optimization-based methods.

While these large models push the state-of-the-art, they also highlight some gaps and considerations in the field. One challenge is data bias and generalization: models like TripoSR, SF3D, and InstantMesh are trained on synthetic data or datasets of common objects (cars, chairs, animals, etc.). When confronted with very different shapes or imaging conditions (for example, images of pollen grains, as in our case), their performance may degrade.

State-of-the-art reconstruction literature is often motivated by the creation of video game assets and movie assets as reflected in its emphasis on high-resolution texturing. The 3D reconstruction of microbiological particles such as pollen grains, however, is only sparsely explored in this field. Each method has different expectations of its input and training regime (e.g., some require the object to roughly fit in a known category, others assume certain lighting or background conditions). These assumptions can become limitations when moving to a new domain such as pollen grains.

3 Models and Evaluation Procedures

In this chapter we present the models we deemed viable for the given task of 3D pollen reconstruction under our constraints of two orthogonal images as an input. In order to get a broad view of different approaches, we chose models ranging from view-synthesis approaches that learn a physical process, models that directly learn to map input images to output volumes as well as a large generalized 3D shape generation model on which we can perform zero-shot evaluation.

First, we will discuss and justify our selected reconstruction models. After the justification we will dive deeper into each selected model. There we will take a deeper look into how each model works and what their input, output and training process looks like. At last, we give an overview what changes we have made to exploit each model to our pollen grain reconstruction use case as well as our hyperparameter settings for reproduction.

3.1 Selection of Models

We select models that (i) cover the major 3D representation families denoted in Section 2.1, (ii) run or train on our available hardware (single GPUs with up to 24 GB VRAM on the FHNW Slurm cluster), and (iii) produce outputs that we can standardize as meshes for a single evaluation pipeline.

Recent state-of-the-art systems, such as InstantMesh [17], SparseFusion [19], and Zero-1-to-3 [18], rely on large diffusion backbones and vision transformers. We attempted to execute these models locally and on the cluster, but they exceeded memory limits. We therefore do not include these models in our thesis.

Selection criteria

We intentionally span five families: (1) silhouette/analytic reconstruction, (2) voxel CNNs, (3) mesh deformation with graph networks, (4) encoder-conditioned radiance fields, and (5) large 3D diffusion priors usable in zero-shot mode. All methods take the same inputs at inference time in our experiments (two orthogonal, posed views). Since their native output representation varies, common evaluation becomes difficult. We perform a few steps to convert all results to a common representation before evaluating them under a unified protocol. We will explain exactly how we achieve this later in this section.

Model	Family	Native output	Training regime
Visual Hull [20]	Silhouette carving (analytic)	Occupancy grid	N/A (no learning)
Pix2Vox [21]	Voxel CNN (multi-view)	Occupancy grid	From scratch
Pixel2Mesh++ [9], [22]	Mesh deformation	Triangle mesh	From scratch
PixelNeRF [23]	Encoder-conditioned NeRF	Density/radiance field	From scratch
Hunyuan3D-2 [14]	3D latent diffusion prior	Triangle mesh	Zero-shot

Table 3.1: Overview of models included in our study. The selection covers different representations under the constraints of our computational limitations.

3.2 Visual Hull

The visual hull [20] reconstructs shape from silhouettes alone. For each orthogonal pollen grain view, we back-project the 2D silhouette into 3D to form a *visual cone*; intersecting these cones yields the largest volume that reproduces all silhouettes. This makes the method an ideal baseline: it is parameter-free, fast, and grounded in the geometry we directly see on pollen grain

scans, so it validates our camera poses and masks and sets a lower bound for learned methods to surpass.

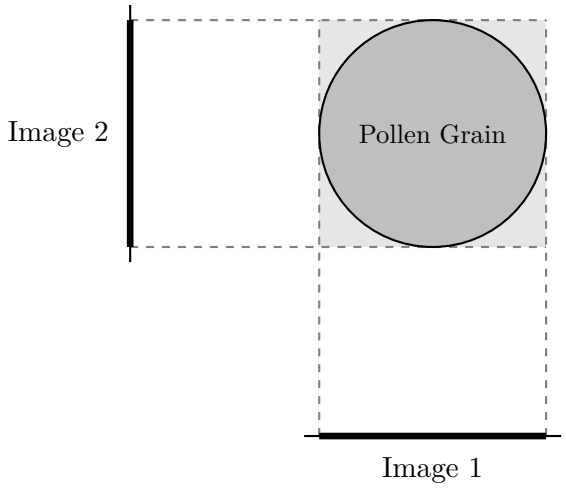


Figure 3.1: Visual hull from two orthogonal, calibrated views. Each silhouette (C_1 and C_2 on Image 1 and 2) back-projects to a cone (or extrusion under orthographic projection); their intersection "carves" the volume $\mathcal{H}$. Light-gray regions illustrate inevitable over-fill where concavities are invisible in silhouettes.

Mathematically, if C_i is the binary silhouette from view i, the visual hull can be described as

$$\mathcal{H} = \bigcap_{i=1}^{N} C_i$$

We adopt orthographic projection to match our preprocessing pipeline, and ultimately the Swisens measurement device. This orthographic projection can be seen on Figure 3.1.

The approach exposes a fundamental limitation that is especially relevant for pollen grains: any concavity that never appears in a silhouette (e.g., an inward groove) is unrecoverable. The visual hull therefore *over-fills* such regions by design. Recognizing this limitation helps us interpret results later: when learned methods outperform the hull on fine depressions, they exploit cues beyond silhouettes.

Practically, most systems compute $\mathcal{H}$ by *voxel carving*: discretize a cubic volume, project each voxel into every view, keep the voxel if and only if all projections fall inside their silhouettes, otherwise carve it away. Early implementations ran this test voxel-by-voxel on CPUs, sometimes accelerating the search along epipolar lines [24]. As GPUs became programmable, researchers mapped silhouettes to textures and carved slice-by-slice at interactive rates [24], [25]. A related family, *space carving*, replaces binary silhouettes with a radiometric test: a voxel survives only if its predicted color agrees across views (photo-consistency), yielding a *photo hull* that lies within the visual hull and can better approach the true surface [26]. We do not rely on photometric tests here because our baseline isolates the information content of silhouettes alone.

3.2.1 Input and Output

We use two orthogonal, calibrated views by default; each view provides a binary foreground mask (silhouette). Our acquisition follows a turntable-like setup where cameras rotate around the global Y-axis at fixed horizontal angles (once at 0° and once at 90° to obtain two orthogonal

views). The method trivially extends to more views (we evaluate up to six evenly spaced views in a later experiments).

The algorithm produces a Boolean occupancy volume on a cubic voxel lattice of resolution R^3. We keep the lattice resolution tied to the input view resolution for reproducibility (R^2). We standardize outputs in the common evaluation pipeline (Section 3.7).

No classical learning or loss function is involved. The procedure deterministically intersects per-view occupancy tests, so there are no trainable parameters or optimizers. The method essentially only translates the binarized silhouettes into a 3D space.

3.2.2 Modifications and Settings

We implement carving as tensor operations suited to modern GPUs. For each view, we transform the voxel lattice into the camera frame using the known horizontal angle about the Y-axis, project voxel centers with an orthographic model, and sample the silhouette at sub-pixel locations via bilinear interpolation. Bilinear sampling avoids aliasing at mask boundaries and keeps the pipeline differentiable. We then apply a hard threshold of 0.5 per view to obtain a binary occupancy test and fuse views with a running logical AND over the lattice. This reproduces the classical definition while remaining fast and autograd-compatible should one wish to integrate it as a prior.

Our default visual hull settings:

- **View set.** Two orthogonal views by default; extended experiments use up to six horizontal angles, evenly spaced around the Y-axis (turntable assumption).
- **Occupancy threshold.** Fixed at 0.5 for all views; a voxel survives only if all per-view samples exceed the threshold.
- **Volume resolution.** Cubic grid of size $R^3 = 32 \times 32 \times 32$; we set R to match the input view resolution of 32×32px. We also added the option of down-sampling the final volume by an integer stride (`step`) for memory/performance trade-offs. Within our experiments however, this always stays at `step`$= 1$.

3.3 Pix2Vox

Pix2Vox reconstructs a 32^3 occupancy volume from single or multiple images using four modules: a shared *encoder–decoder* that predicts a coarse volume for each view, a *context-aware fusion* that scores and fuses those coarse volumes voxel-wise, and a *refiner* that corrects local errors in the fused result [21]. Unlike RNN-based fusion such as 3D-R2N2 [11], which we laid out in Section 2.3, Pix2Vox processes views in parallel, is permutation-invariant with respect to input order, and avoids long-term memory issues [21]. We include it because it (i) natively handles multi-view inputs without camera poses, (ii) learns an implicit shape prior that is valuable on class-specific datasets (pollen), and (iii) offers a strong accuracy-efficiency trade-off among voxel methods [21].

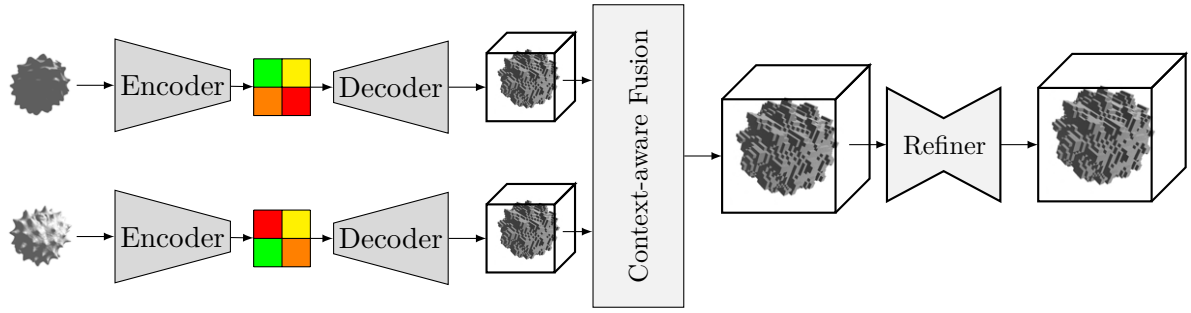


Figure 3.2: Pix2Vox architecture (encoder–decoder, context-aware fusion, refiner). The encoder–decoder produces a coarse volume per view; the fusion module learns voxel-wise weights to combine them; the refiner corrects local artifacts in the fused volume (adapted from Xie, Yao, Sun, *et al.* [21]).

For each input image, a CNN encoder (VGG16 backbone [27]) extracts feature maps; a 3D deconvolutional decoder upsamples them into a coarse 32^3 occupancy prediction. The *context-aware fusion* then computes, for every voxel, a learned score per view from decoder features and uses a softmax-normalized weighting to fuse coarse volumes. This preserves spatial structure and lets the network prefer views where a part is visible (e.g., spines or apertures) [21]. Finally, a U-Net–style [28] 3D *refiner* with skip connections predicts residual corrections that sharpen thin parts and repair under/over-filling [21].

3.3.1 Input, Output, and Training

We feed two orthogonal pollen views by default into the model. Pix2Vox does *not* consume camera intrinsics/extrinsics; it treats images as uncalibrated inputs [21]. Images are resized to 224×224px and passed as 3-channel tensors; for grayscale inputs we replicate the channel and apply ImageNet [29] normalization to match the VGG16 backbone's pretrained weights.

The network predicts a probabilistic occupancy grid of resolution 32^3; we keep this native resolution for fair comparison with the original work and our voxel baseline, the Visual Hull. Any meshing or normalization occurs later in the common evaluation pipeline.

Following Xie, Yao, Sun, *et al.* [21], we optimize voxel-wise binary cross-entropy (BCE) between the predicted occupancy p_i and ground truth gt_i over $N = 32^3$ voxels:

$$\mathcal{L} = \frac{1}{N} \sum_{i=1}^{N} \left[gt_i \log p_i + (1 - gt_i) \log(1 - p_i) \right].$$

This treats reconstruction as dense binary classification over voxels.

We train with the Adam optimizer [30] (β_1=0.9, β_2=0.999) as in the original paper; we use 224×224 inputs and the same output resolution. We inherit Pix2Vox's two-stage schedule (coarse prediction first, then full model with fusion and refiner) to stabilize learning on multi-view inputs [21].

3.3.2 Modifications and Settings

We adopt the *Pix2Vox-A* configuration (heavier decoder, refiner enabled) because it showed better reconstruction capabilities with manageable compute in Xie, Yao, Sun, *et al.* [21].

We follow Xie, Yao, Sun, *et al.* [21] by using a VGG16-BN [27] encoder initialized from ImageNet [29]. To match this backbone, we normalize our images on the fly with ImageNet statistics and, when starting from grayscale pollen renders, replicate the single channel to three channels (The paper also uses 224×224px RGB inputs and a VGG16-based encoder. We keep those defaults.)

Our Merger (context-aware fusion) follows the paper's voxel-wise scoring-and-softmax scheme and replaces any sequence model; it remains permutation-invariant across views [21]. We enable the refiner by default, consistent with Pix2Vox-A, to improve thin structures and local details [21].

We stage training by enabling fusion and refiner after an initial warm-up ("kick-in") so the encoder–decoder first learns stable coarse predictions. This mirrors the two-phase training strategy reported in the paper [21] (coarse stage, then joint training) while adapting the exact switch points to our dataset size.

We use Adam as in the original work and keep BCE as the sole loss. We train for at least 100 epochs before early stopping is performed with a patience of 10 monitoring the validation loss. Hyperparameters such as learning rate, input resolution, voxel resolution, module choices, and optimizer follow Pix2Vox [21].

Two small deviations are worth noting. We added the option to freeze the ImageNet-initialized VGG16 [27] feature extractor to regularize on our domain-specific dataset; the paper does not mandate freezing and later in our experiments we look at the effect of this setting. Per default we fully tune the VGG16 feature extractor, following Xie, Yao, Sun, *et al.* [21].

3.4 Pixel2Mesh++

Pixel2Mesh++ extends Pixel2Mesh from single-view to multi-view mesh generation [9], [22]. The method keeps the "generation by deformation" idea: start from a template mesh (an ellipsoid) and progressively deform it with graph convolutions, then refine it by explicitly reasoning across multiple views. We include Pixel2Mesh++ because it outputs triangle meshes directly (compact, surface-aware, friendly to watertightness checks), leverages multi-view evidence without sequential RNNs, and aligns with pollen morphology, where an ellipsoidal template is a strong initializer. At a high level, the network first predicts a coarse mesh with Pixel2Mesh, then runs a Multi-View Deformation Network (MDN) that samples local deformation hypotheses around each vertex, pools features from all images at those hypotheses, and moves the vertex via a differentiable soft-argmax over hypotheses. This brings classic multi-view reasoning (prefer views where the surface is actually visible) into a learned GCN.

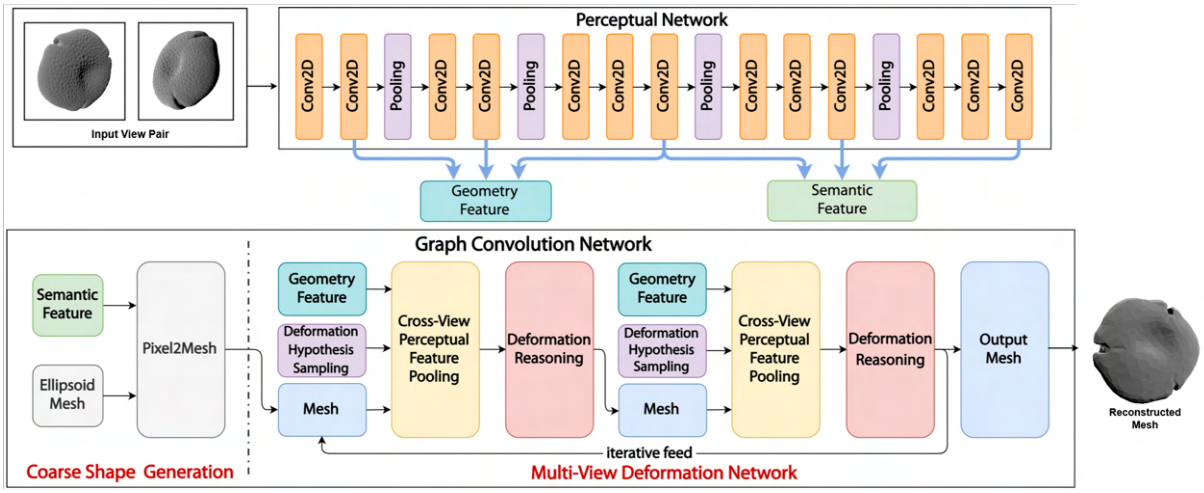


Figure 3.3: Pixel2Mesh++ architecture by Wen, Zhang, Li, *et al.* [9]. The input images and output mesh show a sample from our training partition. The top part shows how we obtain a geometry feature and a semantic feature. The bottom shows the two core reconstruction components, the Coarse Shape Generation (original Pixel2Mesh model [22]) and the new Multi-View Deformation Network [9].

Meshes are graphs: vertices connected by edges form triangles. Pixel2Mesh treats the mesh as a graph and updates vertex positions with graph convolutions that aggregate features from the one-ring neighborhood along edges. In practice, the coarse stage applies three graph-residual blocks with unpooling between blocks to increase resolution; vertex counts evolve from 156 to 628 to 2466, and the network regresses updated coordinates at each stage. This coarse mesh sets up the MDN with reasonable geometry before finer cross-view refinement.

The MDN is the core Pixel2Mesh++ innovation. For every current vertex, it samples 42 candidate displacements on a small icosahedron centered at the vertex, and builds a local hypothesis graph with 43 nodes and 162 edges (icosahedron edges plus connections to the center). Each hypothesis is projected into every input image using the known intrinsics/extrinsics; early CNN feature maps are bilinearly sampled at those 2D locations and then combined across views using order- and count-invariant statistics (mean, max, std). Concatenating these statistics with the 3D coordinates yields a per-hypothesis descriptor. A small GCN scores the hypotheses; a softmax produces weights that define the new vertex location as the weighted sum of hypothesis coordinates (a 3D soft-argmax). Iterating MDN improves thin structures, with performance typically saturating after three iterations [9].

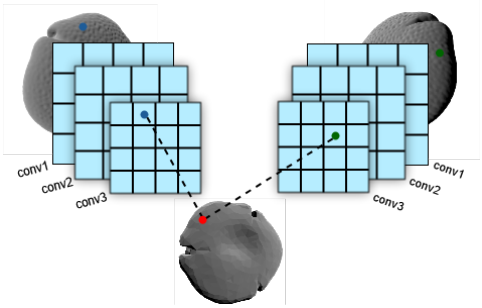


Figure 3.4: Cross-view Perceptual Feature Pooling for a single mesh vertex on a pollen grain. A target vertex on the pollen-grain mesh (shown in red) is projected into each input image using the known camera parameters, yielding image coordinates (blue and green dots in the two orthogonal views).

3.4.1 Input, Output, and Training

We feed two orthogonal, calibrated views by default and vary the number of views in ablations. Pixel2Mesh++ requires camera intrinsics and extrinsics to project vertices and hypotheses into each image for feature pooling (Extrinsic parameters define the camera's position and orientation in the world coordinate system and intrinsic parameters define how the scene is projected onto the image sensor, e.g. the mapping from the scene to pixel coordinates). Inputs are resized to 224×224px and passed through a VGG-style encoder; MDN pools features from earlier layers (higher spatial resolution) to capture local geometry cues. The pipeline is view-order invariant because we aggregate features with statistics rather than concatenation. The model outputs a triangle mesh; We evaluate the finest stage with 2466 vertices.

We supervise with the Pixel2Mesh loss extended by re-sampled Chamfer distance. The loss sums four terms: (i) bidirectional Chamfer distance between point sets sampled from prediction and ground truth; (ii) a normal alignment term that encourages correct face orientation; (iii) Laplacian regularization to stabilize local deformations; (iv) and an edge-length term to discourage long or "flying" edges. Following the paper [9], we uniformly re-sample the predicted mesh using the Ladický re-parameterization trick [31] and compute Chamfer on 4000 sampled points (triangle-area proportional), which reduces artifacts from irregular tessellations and suppresses spikes.

Training proceeds in stages. We first train the coarse Pixel2Mesh path, then enable MDN and allow iterative refinement. In line with the paper, we find three MDN iterations at inference time to be a good accuracy–efficiency trade-off. We also warm up the CNN feature extractor before fine-tuning jointly with the mesh modules to stabilize optimization on our domain.

3.4.2 Modifications and Settings

We use the *official TensorFlow implementation* of Pixel2Mesh++ in its two-stage pipeline: a coarse Pixel2Mesh stage, an intermediate export of coarse predictions, and a fine Multi-View Deformation Network (MDN) stage-matching the authors' setup while keeping the original architecture and losses.

Relative to the original three-view setting, we enable a *configurable* number of input views (typically 2–6) by adapting only the data/pooling wrapper; MDN's statistics pooling remains permutation-invariant and no network changes are introduced.

We expose a *dynamically selectable template prior* (default: ellipsoid) to probe sensitivity to explicit shape priors; when unspecified, the standard ellipsoid is used.

Optimization and loss terms follow the original work; the only change is a *staged learning rate*: `lr`$= 1 \times 10^{-4}$ in the coarse stage and `lr`$= 1 \times 10^{-5}$ in the fine stage, which yielded the best qualitative stability/detail in our setting. We report two training schedules tuned qualitatively: an *augmented* run with 200 coarse epochs followed by 30 fine (MDN) epochs, and a *non-augmented* run with 1000 coarse epochs and 200 fine epochs.

3.5 PixelNeRF

PixelNeRF [23] conditions a neural radiance field on pixel-aligned features from one or more *calibrated* views, enabling reconstruction and novel-view synthesis without per-scene optimization. Unlike explicit surface deformation (Pixel2Mesh++) or discrete occupancy (Pix2Vox), it learns a continuous volumetric field whose density and color are predicted from image features. We include it because it (i) learns a dataset-level prior that generalizes to new specimens and (ii) works with sparse inputs.

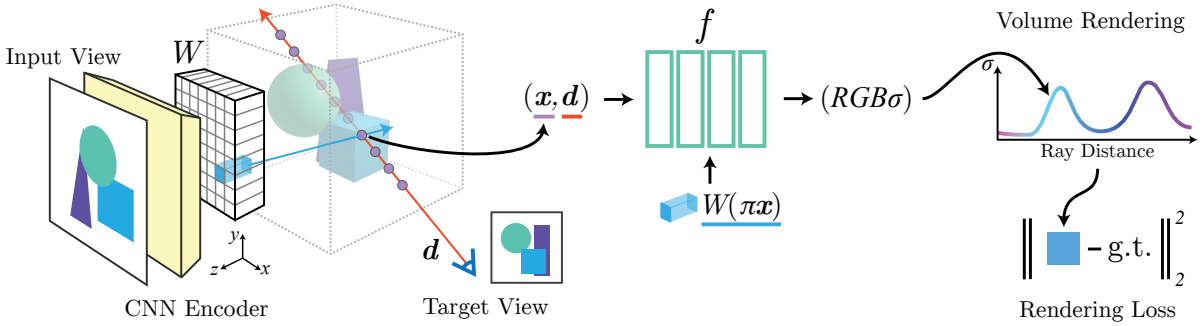


Figure 3.5: Proposed PixelNeRF architecture by [23]. For a point x that lies on a target camera ray with viewing direction d, we first project x into the per-view *2D* feature map W and retrieve the surrounding feature values by bilinear interpolation to obtain its image feature. This feature, together with the spatial coordinates of x and the direction d, is fed into the PixelNeRF network f. The network returns color and density estimates, which are then volume rendered along the ray and compared with the ground-truth pixel value. All coordinates (x,d) are defined in the target camera coordinate system; projections into input views use known intrinsics/extrinsics.

For each input image, a fully convolutional encoder (ResNet-34 [32]) produces multi-scale feature maps. During rendering, 3D points sampled along camera rays are projected into each input view; per-view features are gathered by bilinear interpolation, pooled in an order-invariant way (e.g., mean/max), and concatenated with positional encodings of the 3D point and the ray direction. An MLP predicts density and RGB, and standard volumetric rendering integrates these predictions along the ray to produce pixel colors.

3.5.1 Input, Output, and Training

We train PixelNeRF with *calibrated* inputs (known intrinsics/extrinsics) and between 1–6 views, depending on the experiment A.3. By default, *two* views are sampled uniformly at random from the 128 available training images of the object (per iteration). Images are resized to 256×256 px; grayscale renders are replicated to three channels and normalized with ImageNet statistics to match a ResNet-34 [32] encoder initialized on ImageNet [29]. Feature aggregation across views is permutation- and count-invariant (e.g., mean/max pooling), following Yu, Ye, Tancik, *et al.* [23].

Training minimizes the photometric MSE on both *coarse* and *fine* renderings produced by hierarchical volume rendering [23]. For a batch of rays $\mathcal{R}$,

$$\mathcal{L}_{\text{rgb}}^{(c)} = \frac{1}{|\mathcal{R}|} \sum_{\mathbf{r} \in \mathcal{R}} \left\| C^{(c)}(\mathbf{r}) - C_{gt}(\mathbf{r}) \right\|_2^2, \qquad \mathcal{L}_{\text{rgb}}^{(f)} = \frac{1}{|\mathcal{R}|} \sum_{\mathbf{r} \in \mathcal{R}} \left\| C^{(f)}(\mathbf{r}) - C_{gt}(\mathbf{r}) \right\|_2^2,$$

and the total loss is

$$\mathcal{L} = \lambda_c \, \mathcal{L}_{\text{rgb}}^{(c)} + \lambda_f \, \mathcal{L}_{\text{rgb}}^{(f)} \quad (\lambda_c = \lambda_f = 1).$$

We use stratified sampling for the coarse pass and importance resampling for the fine pass. Unless otherwise noted, defaults are N_c=64 (coarse) and N_f=32 (fine) samples per ray, with rays drawn uniformly across the currently sampled views. Optimization uses Adam [30] for 10^5 iterations (batch size 2, limited by CUDA memory); exact settings (pretraining, positional-encoding frequencies, sample counts) are listed in Appendix A.1.

Inference Phase
During inference, our PixelNeRF implementation operates in a fundamentally different manner compared to training, transitioning from multi-view supervision to sparse-view reconstruction. The inference pipeline is designed to reconstruct complete 3D pollen grain structures from minimal input data while leveraging the learned shape priors acquired during training.

Input Configuration
At inference time, the model accepts between 1-6 input views, with our primary evaluation focusing on 2 orthogonal views positioned 90° apart. These views are resized/normalized using the same statistics applied during training. Each input view requires the corresponding camera-to-world transformation matrix, derived from the spherical camera parameters (θ, ϕ, r) used in our data preprocessing pipeline. If these camera intrinsics and camera poses are not available, there are different strategies to estimate the poses like the estimation they introduced in One-2-3-45 [33].

Feature Extraction and Conditioning
The input images are processed through the same fully convolutional encoder used during training, generating multi-scale feature representations. These features are stored in spatial feature volumes that serve as conditional inputs for the implicit MLP. The permutation-invariant design ensures consistent reconstruction quality regardless of input view ordering, a crucial property for practical deployment scenarios.

Novel View Synthesis
For each target viewpoint, we perform hierarchical volumetric rendering with distance-aware bounds, $t_n = \max(0.1,\, d - 2.0)$ and $t_f = d + 2.0$ (where d is the camera–object distance). We sample N_c=64 coarse points per ray via stratified sampling and resample N_f=32 fine points by importance. Each 3D query point is projected into every input view using known intrinsics and camera-to-world poses to bilinearly sample the per-view *2D feature map*. Multi-view features are aggregated with mean pooling and concatenated with positional encodings of the 3D location and viewing direction $\mathbf{d}$ before being fed to the MLP to predict density σ and color $\mathbf{c}$, which are then volume-rendered along the ray.

Volumetric Rendering
The conditioned MLP predicts color and density values for each sampled point, which are integrated using standard volumetric rendering to produce final pixel values. The learned implicit representation enables rendering at arbitrary resolutions, allowing for detailed geometric analysis through techniques such as marching cubes, which we will discuss later in our Evaluation Pipeline (Section 3.7).

3.5.2 Modifications and Settings

While PixelNeRF shows strong performance in sparse view synthesis and implicit 3D reasoning, we encountered practical limitations during training on our dataset of high-frequency, biologically complex pollen shapes. Most notably, the model occasionally collapsed into predicting entirely black images or degenerate volumes with overly dense or nearly empty radiance fields.

These issues are not unique to our work. In fact, they have been publicly reported by other researchers and users of the official PixelNeRF implementation. For example, community reports

on the official implementation highlight multiple cases where the model's output collapses to entirely black images, with alpha values exceeding valid physical bounds.

To address these instabilities, we introduced the following modifications to the original PixelNeRF training pipeline.

Alpha Sparsity Regularization. We added an explicit regularization term that penalizes the total accumulated opacity α along each ray. This encourages the model to predict sparse, physically plausible densities and prevents over-saturated or empty volumes. For both the coarse and fine rendering branches, we compute the summed opacity and apply weighted penalties:

$$\mathcal{L}_\alpha = \lambda_{\text{coarse}} \cdot \mathbb{E}[\alpha^{(c)}] + \lambda_{\text{fine}} \cdot \mathbb{E}[\alpha^{(f)}]$$

where $\alpha^{(c)}$ and $\alpha^{(f)}$ denote the summed alpha values per ray for the coarse and fine outputs respectively. This regularization stabilizes training and improves volumetric sparsity without compromising reconstruction quality.

Gradient Clipping. We applied targeted gradient clipping to stabilize training. Parameters associated with alpha/density prediction (e.g., sigma layers) are clipped using a stricter norm threshold (ℓ_2 norm ≤ 0.1), while other parameters use a more permissive clipping bound (e.g., ≤ 1.0). This prevents exploding gradients in the density field and ensures that no single batch destabilizes the training process. After qualitatively evaluating multiple threshold configurations and parameter groupings, this asymmetric scheme proved to be the most stable.

Configurable Input Resolution. We expose image resolution as a configuration parameter (propagated through data loading, encoding, ray sampling, and rendering) and train at 256×256; to fit VRAM we reduce the per-GPU batch size accordingly. This improved fine surface detail and morphological fidelity compared to 128×128.

Settings

We use a ResNet-34 image encoder, as in PixelNeRF [23], initialized on ImageNet [29]; inputs are normalized with ImageNet statistics. Grayscale pollen renders are replicated to three channels. All experiments render at 256×256 px and rely on known intrinsics/extrinsics for pixel-aligned feature sampling.

The radiance field is parameterized by a 5-block MLP (hidden width 512) conditioned on pooled multi-view image features, positional encodings of 3D points, and ray directions. We employ hierarchical volume rendering with coarse and fine networks (stratified sampling for the coarse pass and importance resampling for the fine pass). Unless otherwise stated, the default sample counts are N_c=64 (coarse) and N_f=32 (fine), with higher counts used in the augmented configuration.

Optimization uses Adam with a fixed training budget of up to 1×10^5 training iterations; hyperparameters are selected on the validation set. The loss is the RGB photometric MSE applied to both coarse and fine renderings as in Yu, Ye, Tancik, *et al.* [23]. Configuration switches (pretrained vs. from-scratch encoder, global encoder on/off, positional-encoding frequency, and per-stage sample counts). Additional ablations are reported in Appendix A.1.

3.6 Hunyuan3D-2

Hunyuan3D-2 [14] is a two-stage image-conditioned 3D generation system: a shape model first predicts a high-fidelity bare mesh, and a separate texture model later synthesizes view-consistent images for texture baking. Decoupling geometry and appearance simplifies training and supports either generated or hand-crafted meshes. We adopt only the *shape* stage in this work because our evaluation focuses on geometry and our data is grayscale. The shape stack couples a ShapeVAE that compresses meshes into a sequence of latent tokens with a large flow-based diffusion transformer (Hunyuan3D-DiT) that denoises those tokens conditioned on an input image; the decoder then reconstructs a triangle mesh via marching cubes on an SDF predicted by the VAE. The authors publicly release code and pretrained weights for the system, positioning it as an open-source 3D foundation model.

At a high level, the ShapeVAE uses importance sampling to emphasize high-curvature regions during encoding and learns a latent token sequence with variable length (up to 3072 in the released version) to balance efficiency and detail [14]. The DiT then operates in dual- and single-stream transformer blocks: early blocks maintain separate image and shape streams with cross-attention to fuse conditioning, while later blocks refine shape tokens alone. Conditioning uses a strong image encoder (DINOv2 Giant [34]) at a large input size with background removal and centering to improve alignment. The diffusion model is trained with a flow-matching objective and integrated at inference with a first-order ordinary differential equation solver.

3.6.1 Input, Output, and Training

The shape model conditions on a *single* reference image. Internally, a pretrained image encoder extracts tokens from the input view; the DiT predicts a velocity field in latent-token space that transports Gaussian noise toward the ShapeVAE latent for the target object. The VAE decoder then predicts a signed distance field that we convert to a triangle mesh with marching cubes. We do not use the texture stage (Hunyuan3D-Paint), which is designed to generate multi-view images for texture baking; geometry in our experiments comes solely from the shape stage.

Although we do not train Hunyuan3D-2 ourselves, we note the paper's losses for context. The ShapeVAE minimizes SDF reconstruction error with a KL regularizer for the latent (their Eq. (1)), and Hunyuan3D-DiT uses the standard flow-matching ℓ_2 objective to predict velocities along an affine path between noise and data (their Eq. (2)); inference solves the resulting ordinary differential equation from noise to data.

3.6.2 Modifications and Settings

We run the *official* Hunyuan3D-2 multi-view checkpoints in zero-shot mode with no architectural changes. To match our dataset and the paper's conditioning pipeline, we convert grayscale scans to three channels, remove background, center the object, and resize to 518×518 before encoding.

For inference, we use *50 denoising steps* and set the *octree resolution* to *380*; these values gave the best validation performance in our setup. The decoded mesh is passed directly to our common evaluation pipeline A.26.

3.7 Evaluation Pipeline

To draw a conclusion and compare all models in the same way, we built an evaluation pipeline that looks at our test samples. Before running the evaluation pipeline, each model predicts an output on the test sample views. A challenge at this step is that our selected models produce dissimilar outputs that are not directly comparable with each other. As a result of this we inherently introduce inconsistency to the evaluation which makes it difficult to make each model's performance comparable. To mitigate these difficulties in the most optimal way, we included normalization steps to move each model's predictions to a common, more comparable representation, before we calculate reconstruction metrics. In this section we introduce these steps.

3.7.1 Step 1: Bringing all predictions into a common representation

As mentioned, each model first predicts a pollen shape in its native output format, then we convert the output to a common format. In our case we chose the common format to be a triangle mesh. The choice of this common output format emerges naturally, because all samples in our dataset, the 3D Pollen Library [35], are represented as triangle meshes. Calculating the performance metrics directly on the ground truths as they emerge from the 3D Pollen Library [35] ensures that we do not add additional processing that may introduce artificial noise.

Of our selected methods, the Visual Hull, Pix2Vox and PixelNeRF do not directly produce mesh representations. This means that these models are subject to go through this evaluation preprocessing step.

To get a comparable mesh output from these methods, we are using the Marching Cubes algorithm [36], which can be applied both to Voxels and Neural Radiance Fields. The Marching Cubes algorithm by [36] works by overlaying a 3D grid over either our voxel pollen grains or the densities of a NeRF, and extracts a surface by "marching" through the grid cell by cell. For each cube (defined by 8 neighboring sample points), it checks which corners are inside or outside the surface (using a threshold σ), then uses a precomputed lookup table to determine how the surface should cut through that cube. It then places triangle vertices along the cube edges (interpolating positions between inside/outside points) and connects them into triangles. Repeating this for all cubes produces a continuous triangle mesh approximating the surface implied by the scalar field.

For our voxel-based models, we interpret a voxel coordinate to be occupied if the predicted score $p_{i,j,k}$ fulfills $p_{i,j,k} > 0.5$. Thus, for the Marching Cubes algorithm, we set the threshold $\sigma = 0.5$.

For our NeRF-based model, PixelNeRF [23], the densities can not be looked at as binary as its output is a continuous, unbounded (non-negative) quantity representing how much light is attenuated per unit distance at that point in space. We convert the implicit field to an explicit surface using a uniform 256^3 grid over the bounded scene volume. To find an "optimal" threshold σ, we sweep a fixed candidate set $\{0.1, 0.15, 0.2, 0.5, 1.0, 2.0, 5.0, 7.5, 10.0\}$ on the first input scene and keep the chosen value fixed for all other scenes of the same model reconstructions. We select the σ that minimizes the Chamfer Distance on a validation sample. We qualitatively found this to be the best signal to find a suitable value for σ while not leaking to evaluation.

3.7.2 Step 2: Normalizing the predictions

Not every model learns to recover a normalized 3D shape. For example, the output voxel grid of Pix2Vox does not inherently have the same scale as the Pixel2Mesh++ output. This calls for normalization of all predictions.

At this step, after applying Marching Cubes, we only deal with meshes. This means, the normalization step becomes quite trivial. For every vertex of a mesh v_i we give it a new position

$$v_i' = \frac{v_i - \bar{v}}{||v_{\max} - v_{\min}||_2}$$

$\bar{v}$ is the centroid of all vertices in the mesh, $v_{\max}$ and $v_{\min}$ are the coordinate-wise maximum and minimum across all vertices. This has the effect that we center the mesh at the origin centroid (at coordinate $\{0, 0, 0\}$), uniformly scale the mesh so the diagonal of its bounding box is of length 1.

3.7.3 Step 3: Aligning the meshes

The next difficulty lies in the fact that we are handling outputs of objects in 3D-space. Our selected models do not directly learn a back-trackable orientation based on the position and direction of the cameras the input images were recorded with. The only prior information we have in that regard is the camera-to-camera angle, which is orthogonal as per our research goal. In essence, this means we cannot deterministically rotate a predicted shape to the exact same orientation the ground truth lies in. Thus, we have no information of a predicted object's rotation in relation to the ground truth at testing time.

Our first step in our evaluation pipeline is the alignment of the predicted shape to the ground truth.

We achieve this using the Point-to-Plane Iterative Closest Point (ICP) algorithm [37], a variant of the standard ICP algorithm [38]. At a high level, this method repeatedly refines the predicted mesh's position and orientation so that its surface matches the ground truth surface as closely as possible.

The process works as follows:

1. **Surface sampling**: We first sample $N_{\text{points}} = 5000$ vertices from both meshes (predicted and ground truth). These points act as a compact but representative version of each mesh's geometry.
2. **Normal estimation**: For each sampled point, we estimate the direction of the local surface (the normal vector). This tells us how the surface is oriented at that location.
3. **Closest-point matching**: The algorithm pairs each point from the predicted mesh with the nearest point on the ground truth mesh.
4. **Point-to-plane error calculation**: Instead of measuring the straight-line (point-to-point) distance between paired points, Point-to-Plane ICP measures how far the predicted point lies along the normal direction of the ground truth surface. This makes alignment more accurate for smooth surfaces, because it takes surface orientation into account rather than only absolute position.
5. **Rigid transformation update**: The algorithm computes the best rotation and translation that reduce these point-to-plane distances. It applies this transformation to the predicted mesh.
6. **Iteration until convergence**: Steps 3 to 5 repeat until the improvement becomes negligible (at 5% alignment error) or a maximum number of iterations ($N_{\text{max\_iter}} = 10000$) is reached.

After these three steps, our predictions are now ready to get evaluated. In the following section we will establish the metrics we calculate between the ground truths and our newly processed predictions.

3.8 Evaluation Metrics

We recorded numerous 3D metrics but only deemed a handful relevant and common enough among 3D reconstruction literature. More specifically, we opted to consult metrics that were used in our selected models original publications so that we have a rough guideline of the expected results. For example, Chamfer Distance (CD) and F-Score in [9], as well as Intersection-over-Union (IoU) in [21].

We observe that standard shape-generation metrics, such as occupancy measures or point-to-point distances, do not fully capture critical surface characteristics like continuity, smoothness, or higher-order detail. Since there is no widely accepted metric for these aspects in literature, we advocate complementing quantitative scores with thorough qualitative analysis to more accurately assess surface quality.

3.8.1 Symmetric Chamfer Distance (CD)

To quantify the geometric discrepancy between the reconstructed pollen grain and its ground truth model, we follow common practice in 3D shape analysis and employ the symmetric Chamfer distance. This metric is popular in the reconstruction literature, as already seen in [9], [22], because it is (i) invariant to point ordering [39], (ii) robust to moderate sampling noise and occlusions [40], and (iii) empirically correlated with perceptual surface fidelity [40].

Within our evaluation pipeline we obtain $N = M = 5000$ sampled points (i.e. vertices) for every predicted mesh surface S_{pred} and every ground-truth mesh surface S_{gt}.

For every point of one set we locate the nearest point in the opposite set and average these nearest-neighbour distances in both directions. The result is a symmetric, first-order measure of geometric discrepancy. Small values indicate that the two surfaces closely overlap, whereas larger values signal missing or extraneous geometry.

Formally, let

$$S_{\text{pred}} = \{x_i\}_{i=1}^{N} \subset \mathbb{R}^3, \quad S_{\text{gt}} = \{y_i\}_{j=1}^{M} \subset \mathbb{R}^3$$

be the point clouds obtained by uniform sampling of the predicted and ground-truth pollen meshes, respectively. The symmetric CD is

$$\text{CD}(S_{\text{pred}}, S_{\text{gt}}) = \frac{1}{N}\sum_{i=1}^{N} \min_j ||x_i - y_j||_2 + \frac{1}{M}\sum_{j=1}^{M} \min_i ||y_j - x_i||_2.$$

We can look at this metric in two terms. The first term characterizes the coverage error, in essence, how well the prediction reaches every part of the ground truth. The second term, the completeness error, describes how much error the prediction introduces beyond the ground truth.

Because we normalize both meshes to a unit bounding box prior to sampling, the CD values are directly comparable across specimens.

3.8.2 F-Score

The symmetric CD tells us how far surfaces deviate on average, but by construction it can be dominated by a handful of large outliers and is expressed in length units rather than intuitive percentages. To complement CD, we therefore report the F-Score at fixed tolerances τ_n, i.e. the harmonic mean of precision and recall measured on the two point sets of the prediction and ground truth.

To provide this view we compute an F-Score on the same uniformly sampled surface points as established before, S_{pred} and S_{gt}, treating proximity as a "hit" if two points are closer than a predefined tolerance τ.

Because we center and isotropically scale both our ground-truth and predicted meshes so that the diagonal of their joint bounding box equals 1, a distance threshold τ corresponds to $\tau \times 100\%$ of the current grain's bounding-box diagonal. Hence our three defined thresholds

$$\tau_1 = 0.01, \quad \tau_2 = 0.025, \quad \tau_3 = 0.05$$

evaluate geometric agreement at 1%, 2.5% and 5% of the specimen's size. Thus, we achieve a comparability across pollen taxa of different absolute scales.

This means for a given threshold τ we define

$$\text{precision}(\tau) = \frac{1}{N} \sum_{i=1}^{N} 1 \left[\min_j ||x_i - y_i||_2 < \tau \right],$$

$$\text{recall}(\tau) = \frac{1}{M} \sum_{j=1}^{M} 1 \left[\min_i ||y_j - x_i||_2 < \tau \right],$$

$$\text{F-Score} = \frac{2 \cdot \text{precision}(\tau) \cdot \text{recall}(\tau)}{\text{precision}(\tau) + \text{recall}(\tau)}$$

A higher percentage at a given threshold τ corresponds to a better reconstruction.

3.8.3 Intersection over Union (IoU)

The previous metrics operate directly on surface points. To verify that a reconstructed pollen grain also captures the overall volume and silhouette, we compute the IoU after voxelizing both meshes:

1. Enclose the ground-truth and predicted mesh in a common, axis-aligned bounding box.
2. Divide that box into an equal-spaced $32 \times 32 \times 32$ voxel grid.
3. Mark a voxel as occupied if any part of the corresponding mesh lies inside it.

With the IoU we move away from surface-specific metrics and thus, we define

$$V_{\text{pred}}, V_{\text{gt}} \subset \mathbb{Z}^3$$

for both the ground-truth and predicted meshes. After obtaining these index sets of occupied voxels $(V_{\text{pred}}, V_{\text{gt}})$ we can calculate the IoU as

$$\text{IoU}(V_{\text{pred}}, V_{\text{gt}}) = \frac{|V_{\text{pred}} \cap V_{\text{gt}}|}{|V_{\text{pred}} \cup V_{\text{gt}}|}.$$

IoU therefore ranges from 0 (no overlap) to 1 (perfect volumetric match).

4 Data and Preprocessing

The goal of this thesis calls for a comparison of multiple reconstruction approaches. In this section, we introduce our selected pollen dataset. With multiple reconstruction methods that we aim to compare, we also encounter the challenge of processing the data in a way that is digestible for each method. Each model we will look at has unique requirements regarding input images, ground truth shape formats and even meta information such as camera intrinsics. Due to these circumstances, we have chosen against building a monolithic preprocessing pipeline in order to follow each approach's preprocessing in a clear way. Thus, in a second step we present our hierarchical preprocessing strategy.

4.1 Dataset: 3D Pollen Library

We use the publicly available 3D Pollen Library [35]. The 3D Pollen Library dataset comprises high-resolution volumetric image stacks and corresponding triangle surface-mesh files for individual pollen grains. The pollen grains come from a wide range of plants, ranging from grasses, rushes, fruiting plants and herbs to mosses and ferns.

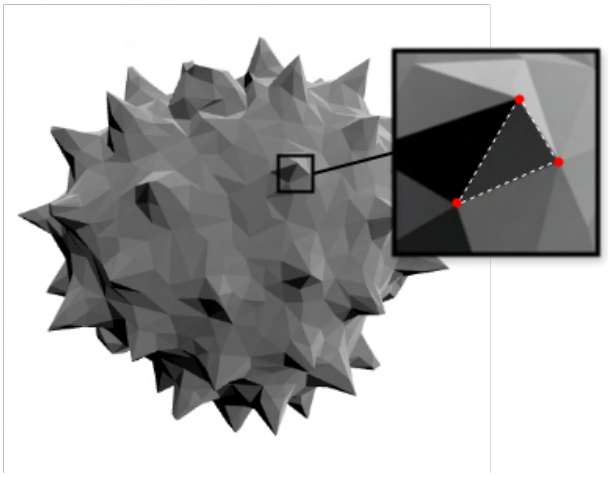


Figure 4.1: Sample of a simplified pollen grain mesh showing the surface made out of triangles. The red dots are vertices, the dashed lines are edges. Together they form a triangle and span a "face".

This 3D Pollen Library dataset obtained meshes of pollen grains by imaging the fine particles via microscopy using an oil-immersion objective to capture the outer shell of pollen (i.e. sporopollenin, the primary component of the exine).

Table 4.1: Summary of the 3D Pollen Library dataset

Dataset	# Samples	# Unique Species
3D Pollen Library	207	201

Figure 4.2: Overview of mesh samples of the 3D Pollen Library dataset. The plot shows how the morphology varies largely across the entire dataset.

4.1.1 Data Partitioning

We split the dataset of 207 samples into 70% training, 15% validation, and 15% testing. The selected dataset shows high sparsity with 207 overall samples of which 201 are unique species. A typical data splitting approach in scenarios with small datasets, as such, would be to apply k-Fold Cross Validation to characterize each model's prediction error. However, we would naturally introduce more bias in every split since every fold would validate on species that have not been learned before. Although the general domain of the dataset stays the same, morphologies differ vastly between species [41], even within the same family (Eurypalynous taxa) [42, p. 13-15]. With the high sparsity of species in mind, we split the 207 samples into the aforementioned splits randomly while making sure that the few species that occur multiple times do not exist in two partitions at the same time. Thus, we avoid leakage of species.

Given the limitations of the selected dataset, we apply data augmentation on different magnitudes on the training split only in order to escape the "one-shot" scenario of the pollen species. We discuss the detailed augmentation strategy within the following Section 4.2.

4.2 Base Preprocessing

Our pollen reconstruction pipeline begins long before any network sees an image or a mesh. The data preprocessing that we present in this thesis follows a hierarchical approach that prepares

Table 4.2: Summary of the data partitioning

Partition	# Samples
Train	144 (70%)
Validation	31 (15%)
Test	32 (15%)

the 3D Pollen Library dataset for training across multiple 3D reconstruction methods. At the foundation of this pipeline lies the Base Preprocessing stage, which performs essential data augmentation and geometric corrections on the raw pollen dataset. This base stage extends our dataset by adding augmentations and converting all pollen meshes to manifold and watertight representations.

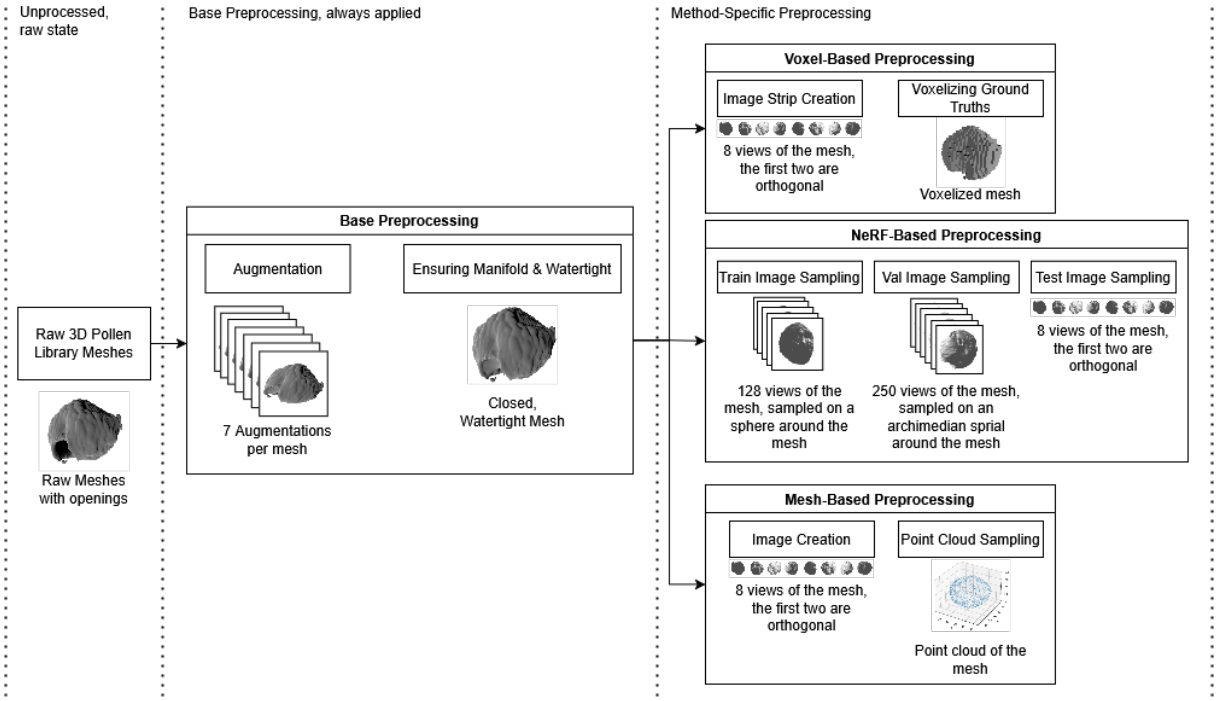


Figure 4.3: Overview of our preprocessing pipeline hierarchy. The raw 3D Pollen Library [35] first gets processed by the Base Preprocessing. After that we preprocess the "base dataset" further, respectively for each overarching reconstruction method.

Building upon this foundation, we implement three specialized and method-specific preprocessing branches as depicted in Figure 4.3. Each preprocessing branch allows for the specific requirements of different model architectures with their 3D representations we have already established in Section 2.1. The voxel-based preprocessing pipeline transforms the base-processed data into the volumetric representations required for training Pix2Vox models. The mesh-based preprocessing branch generates all necessary mesh-specific artifacts needed for Pixel2Mesh++ training, including surface sampling. Finally, the NeRF-based preprocessing pipeline prepares the data for PixelNeRF training by establishing the appropriate scene representations and camera configurations.

4.2.1 Data Augmentation

A benefit of this augmentation strategy is the creation of multiple samples for each pollen species. This allows us to generate training images from different perspectives for each species.

Each mesh is normalized, scaled, and subjected to several distinct nonlinear deformations, modeled using spatial transformations. The strength, orientation, and parameters of each deformation is randomized per sample to maximize variation.

Each transformation is parameterized by a time-scaled intensity factor $t \in [0, 0.4]$, defined per augmentation index i as:

$$t = \frac{i}{N-1} \cdot 0.4$$

where N is the number of augmentation steps. Random seeds and deformation parameters are sampled independently for each mesh to ensure non-deterministic and diverse augmentation results. All augmented meshes are exported in STL format and categorized by deformation type.

We implemented our augmentation strategy to mimic nature's variability of pollen as closely as possible [43]. For this we apply the following seven augmentation strategies:

- **Twisting**. This transformation rotates each vertex around a principal axis (typically the Z-axis), with the rotation angle depending on the vertex height:

  $$T_{\text{twist}}(\mathbf{v}) = R_z(\alpha z) \cdot \mathbf{v}$$

  where R_z is a rotation matrix and $\alpha \sim \mathcal{U}(-a, a)$ is a randomized twist factor.
- **Stretching**. Stretching applies tapering along a randomly chosen axis, elongating one end of the mesh:

  $$T_{\text{stretch}}(x, y, z) = (x, y, z \cdot (1 + \beta z))$$

  where $\beta \sim \mathcal{U}(b_1, b_2)$. Additionally, a noise-based displacement texture adds fine surface detail.
- **Groove**. Groove deformation bends the mesh cylindrically and adds wave-like perturbations:

  $$T_{\text{bend}}(\mathbf{v}) = \mathbf{v} + \delta \cdot \sin(\omega z)$$

  where δ and ω are randomized to simulate grooves of varying depth and frequency.
- **Asymmetry**. This transformation introduces biologically realistic asymmetries using tapering and noise:

  $$T_{\text{asym}}(\mathbf{v}) = \mathbf{v} + \epsilon \cdot \mathcal{N}(\mathbf{v})$$

  where $\mathcal{N}$ is a cloud-based noise texture and $\epsilon \sim \mathcal{U}(e_1, e_2)$ controls the displacement amplitude.
- **Irregular**. Irregular deformation composes a random subset of 2–3 mild transformations (e.g., twist, bend, cast, displace, lattice):

  $$T_{\text{irreg}} = T_1 \circ T_2 \circ \cdots \circ T_n, \quad n \in \{2, 3\}$$

  Each T_i is randomly sampled per mesh and applied with low strength, preserving global shape.
- **Radical Reshape**. This operation combines moderate bending, casting, and lattice-based warping:

  $$T_{\text{radical}} = T_{\text{bend}} \circ T_{\text{cast}} \circ T_{\text{lattice}}$$

Control points in the lattice are perturbed as:

$$\mathbf{p}' = \mathbf{p} + \gamma(1 + \text{dist}(\mathbf{p})) \cdot \text{rand}()$$

where $\gamma \sim \mathcal{U}(g_1, g_2)$ and $\text{rand}() \in [-1, 1]^3$.

- **Combined**. This is the most complex deformation, combining all core types with slightly amplified strength:

$$T_{\text{full}} = T_{\text{twist}} \circ T_{\text{bend}} \circ T_{\text{stretch}} \circ T_{\text{taper}} \circ T_{\text{lattice}}$$

The resulting shapes display high structural variation, making this variant well-suited for robustness training.

We apply the presented augmentations to the train partition. For each of the 7 defined augmentation types, we generate 5 randomized variants per mesh. We apply each variant with random strength parameters to ensure diverse outcomes even within the same deformation category.

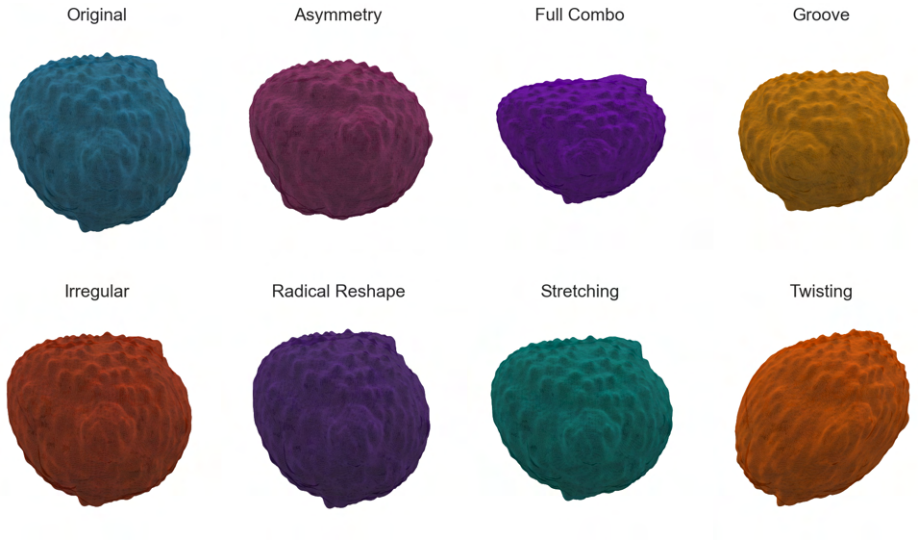


Figure 4.4: An original pollen sample next to our seven different augmentation strategies. Some augmentations may look subtle, the change in 3D space is however quite large, if volumes are overlapped.

Table 4.3: Augmented Mesh Count per Deformation Type

Augmentation Type	**# Generated Meshes**
Twisting	720
Stretching	720
Groove	720
Asymmetry	720
Irregular	720
Radical Reshape	720
Combined	720
Total Augmented Meshes	**5040**

When combined with the original meshes, this produces a significantly enlarged and more variable dataset, which we expect will benefit learning robust implicit shape priors compared to proceeding in a low-data regime.

4.2.2 Manifoldness & Watertightness

As described in Perry, Szeto, Isaacs, *et al.* [35], the dataset's meshes represent only the outer shell of the pollen grains. Thus, most of the meshes follow a hollow non-manifold structure. However, in nature, pollen grains exist in two manifold states: dry and hydrated [42, p. 57]. Dry pollen undergo structural deformation and are subject to infolding when losing water, but in any state, pollen are filled with cytoplasm (gel-like water substance containing proteins). We replicate this structure, by processing each mesh into a manifold shape and making it watertight. We present the mathematical properties of our manifoldness and watertightness preprocessing algorithm in Appendix A.8.2.

4.3 Voxel-Based Preprocessing

To train voxel-grid reconstruction models like Pix2Vox, we convert our normed, smoothed 3D pollen meshes into volumetric occupancy representations. We built this part of the pipeline so it aligns with Pix2Vox's assumptions and architecture but to simultaneously be adapted to be used with our Visual Hull baseline. Among requiring voxelized ground truths, Pix2Vox also requires watertight objects [21]. We already established this within our Base Preprocessing pipeline, as explained in 4.2.2.

4.3.1 Multi-View Image Generation

Pix2Vox may be fed single or multiple views. Our pipeline always generates 8 orthographic views of each mesh, organized as a 1×8 composite image (224px tiles). Cameras are placed at evenly spaced azimuths with constant elevation around the object.

These stereo templates provide the same input format Pix2Vox uses. A fused view-by-view feature volume allow context-aware fusion which later encourages each tile's decoded volume to independently model geometry and then be spatially merged [21]. We will later on discuss the detailed inner workings of this model in 3.3.

4.3.2 Voxelization to Boolean Grid

Each mesh is voxelized into a 32^3 occupancy grid, mirroring Pix2Vox's resolution [21]. The transformation consists of:

- Mapping world space to voxel indices via scaling and translation:

$$v_{\text{voxel}} = S\big(v_{\text{world}}\big) + O, \quad S = \frac{\text{res}/2}{\text{parallel\_scale}}, \quad O = \Big(\tfrac{\text{res}}{2}, \tfrac{\text{res}}{2}, \tfrac{\text{res}}{2}\Big).$$

- Filling interior voxels so that solid pollen grains become dense Boolean cubes, not hollow shells.
- Converting to a $(32, 32, 32)$ binary tensor marking occupied cells.

This explicitly aligns with Pix2Vox's input/output expectations: an occupancy grid per image pair that the refiner merges per-voxel [21]. We introduced the first step mainly to ensure that each voxelized object takes up about the same space as are taken up in the images. This is largely beneficial for the Visual Hull baseline as this method directly predicts voxel grids based on the occupied pixels that can be seen on the input images [20]. The second step is also crucial. When we convert a bare mesh to a voxel occupancy grid, we end up only voxelizing the outer shell of the object, or in palynological terms, the exine. In our Base Preprocessing 4.2.2 we solely ensured watertightness but not solidity. Meshes naturally only describe an outer shell of

an object, represented as a graph. When we move to voxel space however, we can represent solid objects by filling the inner space with voxels, which naturally is a more closely-aligning representation to the pollen grains observed in the real world.

4.4 Mesh-Based Preprocessing

The mesh-based preprocessing stage converts raw STL scans of pollen grains by [35] into a training package that satisfies the architectural and learning assumptions of modern mesh-deformation networks, most notably the Pixel2Mesh++ family, which we will introduce in detail later. In this section we motivate each step and relate it to the generic requirements that mesh-centric methods impose.

4.4.1 Normalization to a Canonical Space

Mesh-deformation networks expect the initial shape to occupy a consistent, scale-free reference frame so that the graph convolutions learned on one object transfer to another. Our script therefore

- centers every mesh at the origin and
- scales it to fit inside the unit sphere $\max ||v|| \approx 1$.

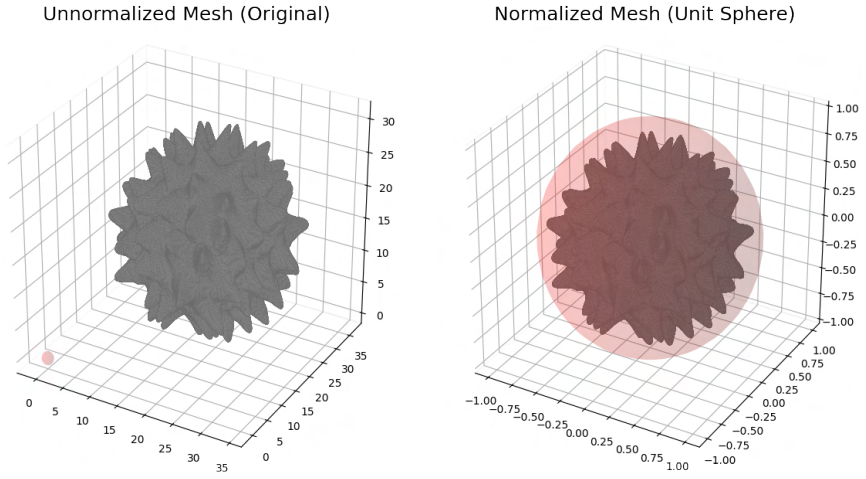


Figure 4.5: Comparison of the original pollen mesh (left) and the normalized mesh (right). The normalization centers the mesh and scales it to fit the unit sphere (shown in red, transparent for left and right).

Pixel2Mesh++ uses the same assumption by starting from a template ellipsoid that has unit extent and zero mean, and then learning vertex offsets relative to that frame [9]. Aligning the pollen meshes up-front lets us re-use the same deformation filters irrespective of natural size variation between small and large pollen.

4.4.2 Topology Simplification

Graph-based models operate on vertices; GPU memory therefore scales with vertex count. The relevant implementation for us, Pixel2Mesh++ [9], trains on a fixed 2466-vertex template and refines it iteratively. To stay in a comparable regime we run quadric decimation (with a target of $20,000$ faces) and verify that the vertex set spans a full 3D sub-space (rank $= 3$). This removes degenerate connected components or flattened scans that would break the Graph Convolutional

Network (GCN) assumption of manifold-like neighbourhoods we already introduced in Section 4.2.2.

Normals are re-computed and L_2-normalized because surface-based losses (e.g. a normal-consistency term or a shaded reconstruction loss) rely on unit normals.

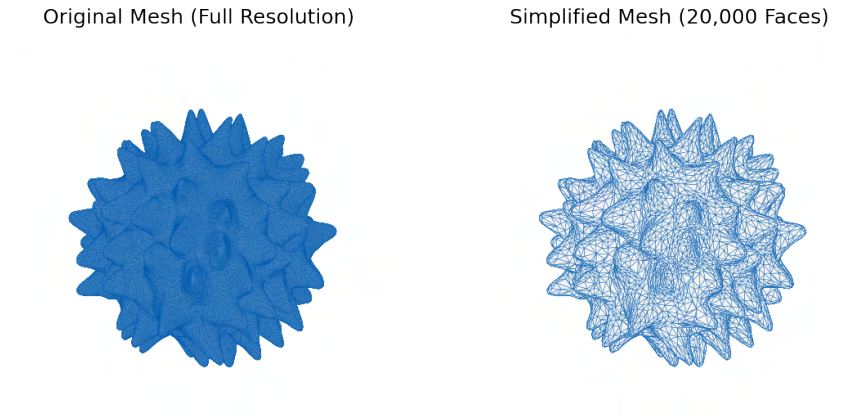


Figure 4.6: Comparison of the original pollen mesh (left) and the simplified mesh (right) after quadric decimation to $\approx 20,000$ faces. The simplified mesh still shows all topological complexities.

4.4.3 Vertex Set as Point Cloud

Mesh-deformation methods like Pixel2Mesh++ do not supervise directly on faces; instead, they compare point samples from the predicted surface to ground truth. Pixel2Mesh++ even re-samples $4,000$ uniform points for its Chamfer loss to avoid bias from irregular triangulations. Rather than sampling stochastically at preload time, we expose the entire vertex set. This:

- preserves surface resolution for later uniform sampling in the training loop,
- lets us compute analytical normals, and
- guarantees deterministic reproducibility across random seeds.

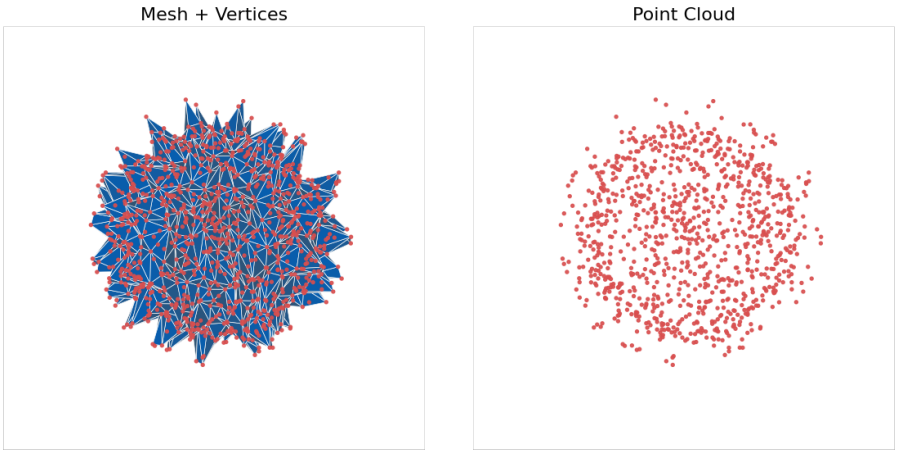


Figure 4.7: The vertices (red points) in our meshes get extracted, forming a point cloud (right) which Mesh-Based models, such as Pixel2Mesh++ [9], [22], can directly optimize and learn from.

4.4.4 Multi-View Rendering with Known Poses

A hallmark of Pixel2Mesh++ is cross-view perceptual feature pooling: each vertex projects into multiple RGB images, where early-layer CNN features are bilinearly sampled and aggregated. That operation requires:

- RGB images with synchronised intrinsics/extrinsics, and
- a vertex-to-pixel mapping that is never ambiguous.

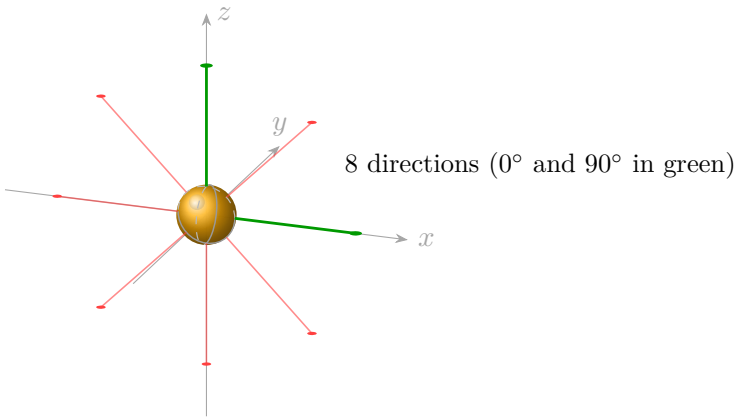


Figure 4.8: Eight evenly spaced view directions on the XZ-plane (y-up). The central sphere (pollen grain) is rendered with volumetric-looking shading and great-circle hints; the two orthogonal base views at 0° and 90° are highlighted in green; the remaining directions are shown in red.

Our renderer produces eight 224×224 images per grain from fixed azimuth/elevation pairs (0°, 90°, 180°, 270°, 45°, 135°, 225°, 315°) and records azimuth, elevation, roll, radius and field-of-view for each view. For our base case we use the first two images, that stand orthogonal to each other. Because the camera always targets the origin (courtesy of the earlier normalization) these parameters are sufficient for the later projection layer. The white background emulates the studio setting assumed by most pretrained 2D backbones.

4.5 NeRF-Based Preprocessing

With mesh-centric approaches handled in the previous section, we now turn to the data pipeline that serves Neural Radiance Field (NeRF) methods. Whereas mesh-deformation networks ingest explicit surface samples, NeRFs learn an implicit volumetric function from multi-view color rays and therefore impose very different requirements on the training corpus. The pipeline described below was designed with these constraints in mind and follows the principles laid out in Yu, Ye, Tancik, *et al.* [23].

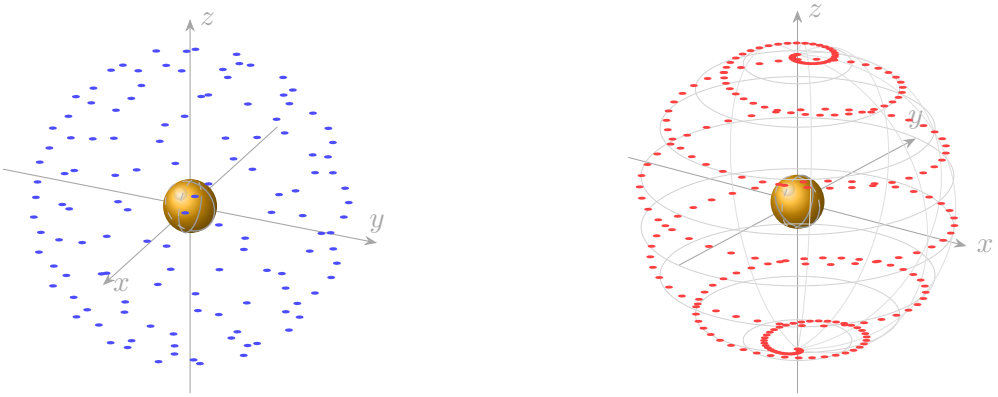
4.5.1 Canonical Scale and Origin

We first translate every mesh to the origin and scale it to fit inside the unit sphere. This echoes the normalization already performed for Pixel2Mesh++ (previous preprocessing section) but with a different motivation. PixelNeRF predicts the scene in the camera's own view space and explicitly avoids a global canonical pose [23] whereas for mesh graph convolutions, such as in Pixel2Mesh [9], assume a fixed starting mesh. Thus, while the math behind our normalization is identical in both the Mesh-Based and NeRF-Based Preprocessing, the semantics differ: the mesh pipeline needs a shared frame; the NeRF pipeline only needs a consistent scale so that ray-sampling parameters generalise across objects and numerical stability is encouraged.

4.5.2 Viewpoint Sampling Strategy

Unlike classical deep learning models that assume data samples across partitions (train, validation and test) to be of the same format, the PixelNeRF method requires data to be sampled differently. A known, common challenge of reconstruction methods such as NeRF is that such methods rely on having an abundance of views to learn from. We discuss the exact implications and what this means in the context of PixelNeRF later.

The original PixelNeRF paper trained the model on ShapeNet ([10]). It contains objects that nearly all have a canonical direction. For example, cars always have an upright position, standing on four wheels. To reconstruct a car, there are never views used from underneath the car. Thus, the viewpoints that have been sampled around the ShapeNet objects have been sampled in a single hemisphere around the object [23]. This case is however not practical for pollen grains. Pollen grains are approximately radially symmetric, so there is no "bottom" we need to hide. To approximate the real-world scenario of pollen getting recorded while airborne, they can be rotated in any direction. As a result of this directional ambiguity, we opted to capture train images on a full-sphere around each mesh. We uniformly sample 128 images per mesh instead of just 50 (like proposed by Yu, Ye, Tancik, *et al.* [23]). This way we can maximize directional coverage.



128 sampled views (train) 250 spiral views (validation)

Figure 4.9: Sampling strategies. Left: 128 training views distributed by spherical Fibonacci sampling (uniform over the sphere). Right: 250 validation views following a spherical spiral from south to north pole. In both panels, the central sphere represents a pollen grain and the blue and red points the position of the camera, directed towards the pollen grain.

For the validation partition we follow the original PixelNeRF implementation [23]. The model gets validated with the full-reference metric Peak Signal-to-Noise Ratio (PSNR). In essence, a view of the reconstructed neural radiance field gets compared to a real image, showing a view at roughly the same camera direction. For this, we sample 250 views on an Archimedean spiral around the ground truth mesh.

Finally, to conform with the goal of reconstructing a grain with a sparse number of input views, we sample the test views (inference views) around the Y-axis (on the XZ-plane). Also here, we sample these views at the same fixed rotations mentioned in 4.4.4, where the first two views are guaranteed to be orthogonal. Both first two views are mainly used later on in the experiments. The additional views are used for our Number-of-Views experiment 5.5, showing the performance when more than just two orthogonal views are available.

As for the dimensions we found a more detailed resolution of 256×256 pixels beneficial during preliminary train runs. Therefore, each view, across all partitions is represented in this dimension.

Table 4.4: Extrinsics/sampling by split (OpenCV cam2world; camera looks along $+z$).

Split	Sampling rule	# views	Near/Far per view
Train	Uniform on sphere	128	near $= \max(0.1,\ d - 2.0)$, far $= d + 2.0$
Val	Spherical spiral (Archimedean)	250	same
Test	"Orthogonal" ring (8 directions)	8	same

Notes. Meshes are centered and scaled to unit radius; all cameras lie on a sphere of radius 2.0.

4.5.3 Per-view Calibration

Next to each 2D-image representation, NeRF-based methods also need the following three camera intrinsics per-image:

- 4×4 **Camera-to-World Matrix**. The camera-to-world matrix transforms points from camera coordinates (local camera space) to world coordinates. It defines the camera's position and orientation in the 3D space of the scene. PixelNeRF explicitly uses camera transformations to map query points into input view spaces when extracting corresponding features from multiple views. More specifically, it uses the camera-to-world matrix to transform query points into the coordinate systems of input views for multi-view aggregation [23]. Without this matrix, the model cannot correctly associate image pixels with locations in the 3D scene.
- **Near/Far bounds t_n^i and t_f^i**. These parameters represent the near and far clipping planes along a camera ray. They limit the range of depth along each ray, essentially defining the region of interest or "viewing frustum" of the scene [13], [23]. The near and far bounds help the model efficiently sample points along camera rays. Without these bounds, the model would have to handle infinitely long rays, making training and rendering impractical and inefficient. Additionally, these bounds help normalize and simplify the sampling process, especially when converting depth to disparity (inverse depth), which is numerically stable and effective for representing distant objects [13].
- **Shared Focal Length and Principal Point**. The Focal Length (f_{cam}) defines the camera's field of view and scale at which objects appear in the image. The Principal Point represents the offset of the camera sensor center relative to the image center. This determines the exact optical center of the camera. The focal length and principal point are essential for the accurate projection and unprojection of points between 3D space and 2D image coordinates. They determine how a 3D point is mapped onto the camera image plane. This known parameter is meaningful for correct ray direction calculations [13]. These intrinsics help describe the correct projections when bilinearly interpolating features from feature volumes generated by CNN encoders as used in Yu, Ye, Tancik, *et al.* [23].

5 Experiments and Results

In this section, we define and present our experiments that aim to answer our research questions. Where applicable, we compare the selected models together in each experiment. Each experiment subsection first defines the goal of the experiment, then lists the models in their setups, and presents the results in a comparative way: once quantitatively and once qualitatively. For the quantitative part we consult the metrics introduced in Section 3.8; Chamfer Distance (absolute), F-Scores (percentage) and intersection-over-union (percentage). At the end of each section, we conclude with insights for all setups in each respective experiment.

RQ	Experiments	Models	Dataset split
RQ1	5.1, 5.2, 5.3	all	test (3D Pollen Library)
RQ2	5.6	Pixel2Mesh++	test (3D Pollen Library)
RQ3	5.5	all (as applicable)	test (3D Pollen Library)
RQ4	5.4	Pix2Vox/PixelNeRF/Pixel2Mesh++	Holographic dataset

Table 5.1: Overview of experiments and evaluation settings by research question (RQ).

5.1 Fine-Tuning / From Scratch Training on 3D Pollen Library

In this experiment, we examine how pretrained encoder-decoder modules affect the training and performance of Pix2Vox [21], PixelNeRF [23] and Pixel2Mesh++ [9] on our 3D Pollen Dataset [35]. We isolate the experiment to use weights trained on the ShapeNet [10] dataset in order to have a better foundation of comparison. We built up our preprocessing to produce pollen images that are closely aligned with the representation of the ShapeNet images. They show our pollen grains as single-colored objects with white background. With these measures we aim to establish transferability of the pretrained weights to our dataset.

Experiment Settings

1. **Visual Hull**: We take the Visual Hull baseline into our consultation without any "state" as this method has no tunable parameters. In its default state ("None"), it takes two orthogonal images as an input and predicts a 32^3 voxel grid.
2. **Pix2Vox**: Within this experiment we set up three configurations for the Pix2Vox model. The "Scratch" state initializes the model with randomly initialized weights for all modules, except for the VGG-16 encoder, which always initializes the pretrained ImageNet V1 weights [29]. The Fine-Tuning ("FT") state initializes the model with ShapeNet [10] pretrained weights across the entire model and then tune all trainable parameters. Lastly, the "Frozen" state describes the experiment setting in which we freeze the feature extraction part of Pix2Vox on the pretrained ShapeNet [10] weights. Thus, both the encoder and decoder are frozen and the Merger and Refiner are tuned.
3. **PixelNeRF**: For the PixelNeRF model we consult the same states as in the Pix2Vox settings. First, we set up the "Scratch" experiment to train the PixelNeRF model entirely with randomly initialized weights across all modules. In the FT state, we initialize all sub-modules with pretrained ShapeNet [10] weights before training the entire model on the 3D Pollen Library [35] dataset. In the "Frozen" state we initialize the model the same way as in the "FT" state with pretrained weights. We, however, only tune the NeRF network, the Encoder stays frozen.

4. **Pixel2Mesh++**: At last, the Pixel2Mesh++ model gets initialized fully with random weights in the "Scratch" state. In the Fine-Tuning state "FT" it also initializes all sub-modules with ShapeNet weights [10] before we start tuning the complete model. Lastly, within the "Frozen" state, we initialize the model as in the "FT" state but freeze the encoder part of the model before training. Thus, we only tune the Graph Deformation Network.

Quantitative Results

Model	State	Chamfer	F-Score			IoU
			$\tau_1 = 1\%$	$\tau_2 = 2.5\%$	$\tau_3 = 5\%$	
Visual Hull	None	0.091 ± 0.025	10.92 ± 4.0	41.6 ± 10.2	66.4 ± 12.1	63.7 ± 9.1
Pix2Vox	Scratch	0.063 ± 0.019	14.9 ± 4.7	56.9 ± 13.8	81.9 ± 11.8	$\underline{72.3 \pm 11.0}$
	FT	0.060 ± 0.021	16.7 ± 6.1	60.3 ± 15.1	84.8 ± 12.1	71.4 ± 17.6
	Frozen	$\underline{0.057 \pm 0.017}$	$\underline{18.0 \pm 6.4}$	$\underline{61.9 \pm 15.0}$	$\underline{85.1 \pm 10.0}$	67.8 ± 14.0
PixelNeRF	Scratch	0.045 ± 0.018	$\mathbf{26.2 \pm 11.5}$	$\mathbf{75.9 \pm 16.6}$	90.1 ± 10.7	$\mathbf{82.6 \pm 8.3}$
	FT	$\mathbf{0.045 \pm 0.017}$	25.7 ± 8.8	75.6 ± 15.3	$\mathbf{90.4 \pm 9.7}$	82.4 ± 7.1
	Frozen	0.059 ± 0.015	17.6 ± 6.6	62.3 ± 9.2	82.8 ± 9.7	79.2 ± 7.8
Pixel2Mesh++	Scratch	0.052 ± 0.023	21.3 ± 9.0	$\underline{69.1 \pm 15.6}$	88.5 ± 12.5	40.6 ± 23.4
	FT	0.052 ± 0.022	21.6 ± 9.6	68.7 ± 15.5	88.0 ± 12.0	$\underline{77.0 \pm 11.0}$
	Frozen	$\underline{0.051 \pm 0.025}$	$\underline{21.6 \pm 9.1}$	68.8 ± 15.9	$\underline{88.7 \pm 14.0}$	76.3 ± 11.8

Table 5.2: Evaluation metrics of models under different transfer learning techniques. The state "Scratch" means we trained the model entirely without ShapeNet pretrained weights. "FT" means we Fine-Tuned the model on ShapeNet pretrained weights and "Frozen" describes the setting in which we tuned the models but left the encoder frozen. The best scores within each model are underlined, the global best metric across models is bold.

Qualitative Results

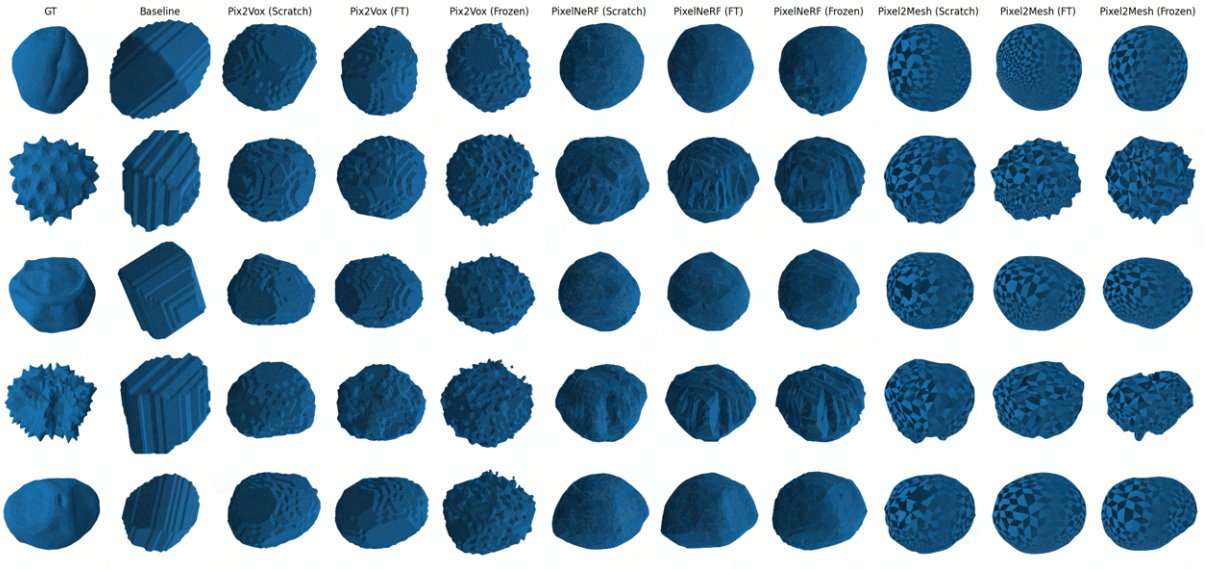


Figure 5.1: Predictions of five test samples showing the ground truth (GT), the Visual hull baseline and the three explored models, each with their three transfer learning settings. The meshes were rotated according to the highest ICP overlap between each prediction and ground truth.

Insights

1. **Pix2Vox**: The metrics show that the Pix2Vox model benefits from transfer learning. Even though, the standard deviation of the metrics is quite large, the mean metrics values show a consistent ordering across the metrics. The model performs best with pretrained weights initialized and then fully fine-tuned (FT setting). The pattern only deviates in the IoU, though, not by a large margin but with a high standard deviation. The qualitative results show that the Pix2Vox model captures the general shape of a pollen in its fully fine-tuned state the best. The Frozen setting shows the most noise/floating vertices in the reconstructions.
2. **PixelNeRF**: Within the explored PixelNeRF settings, the metrics seem to be less clearly separated. The Scratch setting and fully fine-tuned setting (FT) show comparable results, indicating that the model does not clearly benefit from pretrained weights. On the other hand, the experiment setting "Frozen" shows a clearer decrease in reconstruction ability, underlining the inability to benefit significantly from pretrained weights. Additionally, it shows that the model does benefit from more tunable parameters, rather than keeping the encoder frozen. In the qualitative view, the model shows comparable results for all samples. Even though, the "Frozen" setting performed slightly worse, the qualitative reconstructions look comparable to FT and Scratch. The fourth row in Figure 5.1 also shows that the model is able to reconstruct the general geometry in a more detailed way. The crease, facing towards the front in the ground truth, is also reconstructed in all PixelNeRF settings.
3. **Pixel2Mesh++**: In the Pixel2Mesh++ experiments the metrics do not show large differences within all three concluded settings. Similarly to the PixelNeRF experiments, the Pixel2Mesh++ model seems to not clearly benefit from pretrained weights when it comes to our surface metrics. When taking a closer look at the results, the IoU however shows a clear difference. The pretrained weights seem to have given the model a better reconstruction ability in terms of volume. Qualitatively, the reconstructed samples are also comparable within the same model. The spiky Oxeye Daisy pollen grain (second row from the top)

in Figure 5.1 also shows a spiky reconstruction. The Scratch setting, however, does show less detail in that sample which is a worse performance than the other two candidates and consistent with the worse IoU performance.

The models outperform the Visual Hull baseline in any setting. The results however are not clearly significant as the recorded scores show a high standard deviation. This is an expected result as the test partition contains a variety of randomly sampled species that have not been seen before. A closer look at the qualitative results shows similar patterns as recorded with the metrics.

5.2 Data Augmentation

In this second experiment our goal is to look at the impact our base dataset has and whether adding augmentations, as described in Section 4.2.1, improves the model performance in a substantial way. To isolate the effects the augmentation has, we try to move each model into a comparable starting point by initializing them with the corresponding pretrained weights we already described in the settings of the first experiment 5.1.

Quantitative Results

Model	Dataset	Chamfer	F-Score			IoU
			$\tau_1 = 1\%$	$\tau_2 = 2.5\%$	$\tau_3 = 5\%$	
Visual Hull	None	0.091 ± 0.025	10.92 ± 4.0	41.6 ± 10.2	66.4 ± 12.1	63.7 ± 9.1
Pix2Vox	Base	0.068 ± 0.022	15.4 ± 5.7	55.9 ± 13.3	80.6 ± 11.9	61.1 ± 20
	Augmented	$\underline{0.060 \pm 0.021}$	$\underline{16.7 \pm 6.1}$	$\underline{60.3 \pm 15.1}$	$\underline{84.8 \pm 12.1}$	$\underline{71.4 \pm 17.6}$
PixelNeRF	Base	0.046 ± 0.019	25.9 ± 10.4	75.2 ± 15.4	89.6 ± 10.6	81.7 ± 8.3
	Augmented	$\mathbf{0.043 \pm 0.013}$	$\mathbf{26.0 \pm 8.9}$	$\mathbf{76.2 \pm 12.3}$	$\mathbf{90.9 \pm 8.3}$	$\mathbf{82.8 \pm 6.5}$
Pixel2Mesh++	Base	0.060 ± 0.013	15.2 ± 3.3	58.0 ± 8.4	83.6 ± 8.0	35.2 ± 9.3
	Augmented	$\underline{0.052 \pm 0.023}$	$\underline{21.3 \pm 9.0}$	$\underline{69.1 \pm 15.6}$	$\underline{88.5 \pm 12.5}$	$\underline{40.6 \pm 23.4}$

Table 5.3: Evaluation metrics of models compared by base dataset training and augmented dataset training. The best scores within each model are underlined, the global best metric across models is bold.

Qualitative Results

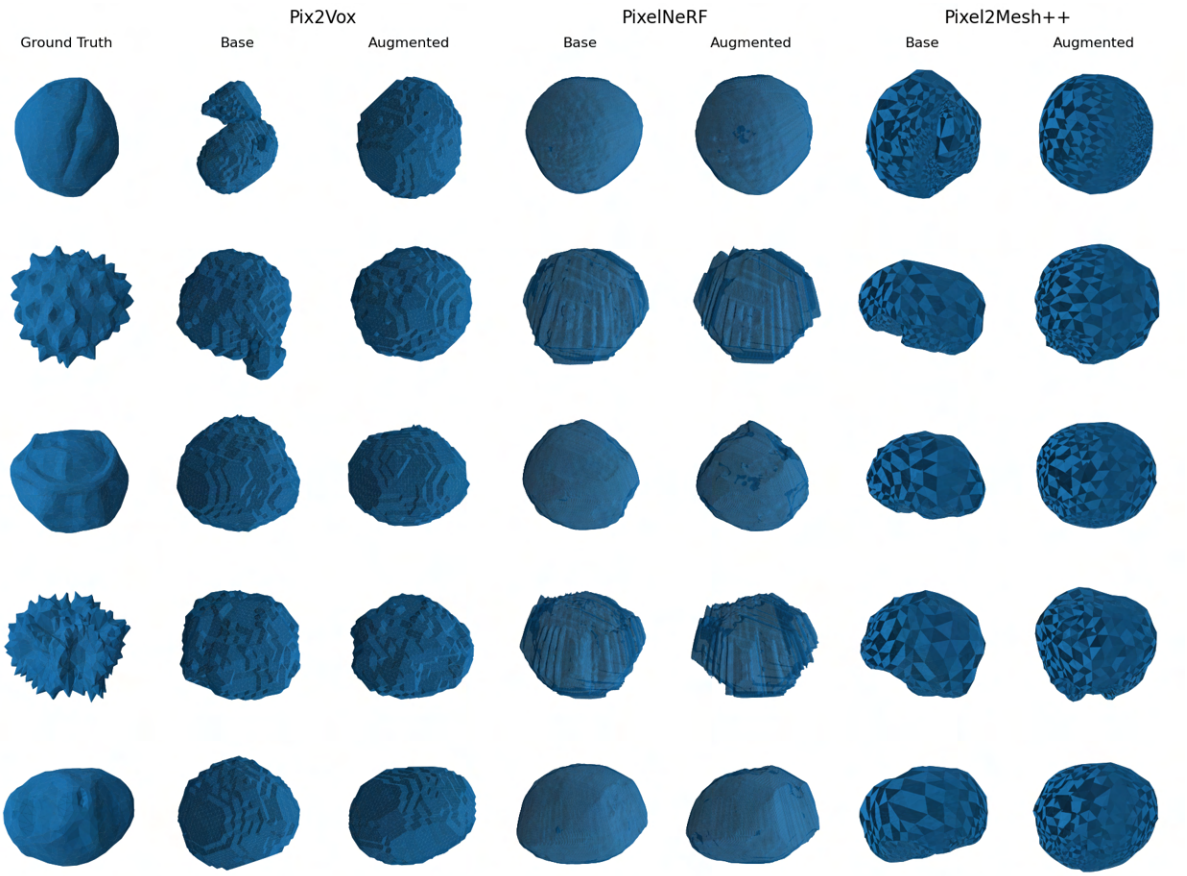


Figure 5.2: Predictions of five test samples showing the qualitative performance for the Pix2Vox, PixelNeRF and Pixel2Mesh++ models when trained on the base dataset versus the augmented dataset. More detailed plots can be found in Appendix A.2.

Insights

1. **Pix2Vox**: The metrics and qualitative comparison demonstrate that Pix2Vox does benefit from being trained on the larger augmented dataset. Within the qualitative results we can see that, especially within the first two meshes, the model struggled with artifacts and shape inconsistencies when trained on the base dataset. Generally, our five sample's shape was better captured on the Pix2Vox version trained on the augmented version. Thus, the Pix2Vox model demonstrates quite a robust improvement in capturing the correct shape, however, it still shows difficulties to reconstruct the smaller details such as spikes or crevices.
2. **PixelNeRF**: The metrics and qualitative samples do not show a significant improvement. In the original paper, PixelNeRF was stated to be a system that requires a substantial amount of data for effective training [23]. This requirement is attributed to the fact that PixelNeRF learns an implicit prior, which necessitates the acquisition of a significant volume of data for accurate prediction. The non-substantial improvements can mostly be attributed to the way PixelNeRF learns. As we have already pointed out in Section 3.5, the PixelNeRF model does not directly learn shapes, but rather it learns the to map any 3D point and viewing direction in a scene to its emitted color and volume density. Our results show that the model is much less dependent on seeing different shapes on its 128

training images. It is however much more dependent on having 128 images to map these density points [23].

3. **Pixel2Mesh++**: The results of Pixel2Mesh++ are more difficult to interpret. The metrics show a substantial improvement when trained on our augmented dataset. However, the qualitative improvements look less substantial and are more challenging to interpret due to the "checkered" prior of Wen, Zhang, Li, *et al.* [9]. To assess the enhancement of the shape across all axes, it is necessary to observe the model from multiple perspectives.

The augmentation hints at some advancements in the Pix2Vox model. The PixelNeRF and Pixel2Mesh++ model mainly improved in the quantitative view of our metrics, less so in the qualitative part. Across our models the augmentation still shows to broadly improve the performance, and thus, we deem it worthwhile to train our selected models on the larger, augmented dataset.

5.3 Zero-Shot Generalization to Pollen Reconstruction using Hunyuan3D-2

As we also look at the larger reconstruction model of Hunyuan3D-2 in this thesis, we introduce this experiment to show its capabilities when used in a zero-shot setting. Thus, we do not train this model on any pollen data, but rather directly perform inference on our test partition on the model, as Zhao, Lai, Lin, *et al.* [14] published it.

Experiment Settings

We use the official multi-view shape-generation pipeline[1] and load its FP16 checkpoint to keep the entire model within a single 16 GB GPU. FlashVDM acceleration is enabled, although we deliberately run a high-quality setting with `num_inference_steps = 50` flow-matching updates. The signed-distance field is decoded on an `octree_resolution = 380` grid, approximately 3.4× denser than the 256 default, to capture the fine-scale ornamentation of pollen grains. To keep the peak decoder memory below 23 GB, we split the volume into `num_chunks = 20 000`. A fixed generator seed (`torch.manual_seed(12345)`) guarantees experiment reproducibility, and we request a triangle mesh via `output_type='trimesh'` for subsequent Chamfer, F-score, and IoU evaluation. All other hyperparameters follow the repository defaults.

Quantitative Results

Model	Views	Chamfer	F-Score			IoU
			$\tau_1 = 1\%$	$\tau_2 = 2.5\%$	$\tau_3 = 5\%$	
Hunyuan3D	2	$\mathbf{0.0432 \pm 0.0126}$	$\mathbf{25.93 \pm 8.69}$	$\mathbf{75.18 \pm 12.47}$	$\mathbf{91.15 \pm 7.80}$	$\mathbf{79.05 \pm 9.58}$
	3	0.0477 ± 0.0148	22.68 ± 10.09	69.68 ± 14.38	89.30 ± 8.66	76.02 ± 10.38
	4	0.0443 ± 0.0129	25.72 ± 9.81	73.55 ± 12.72	90.50 ± 7.63	77.89 ± 9.90

Table 5.4: Evaluation metrics for all Hunyuan3D-2 variants on the augmented 3D Pollen Library dataset. The global best metrics are bold.

[1] https://huggingface.co/tencent/Hunyuan3D-2mv

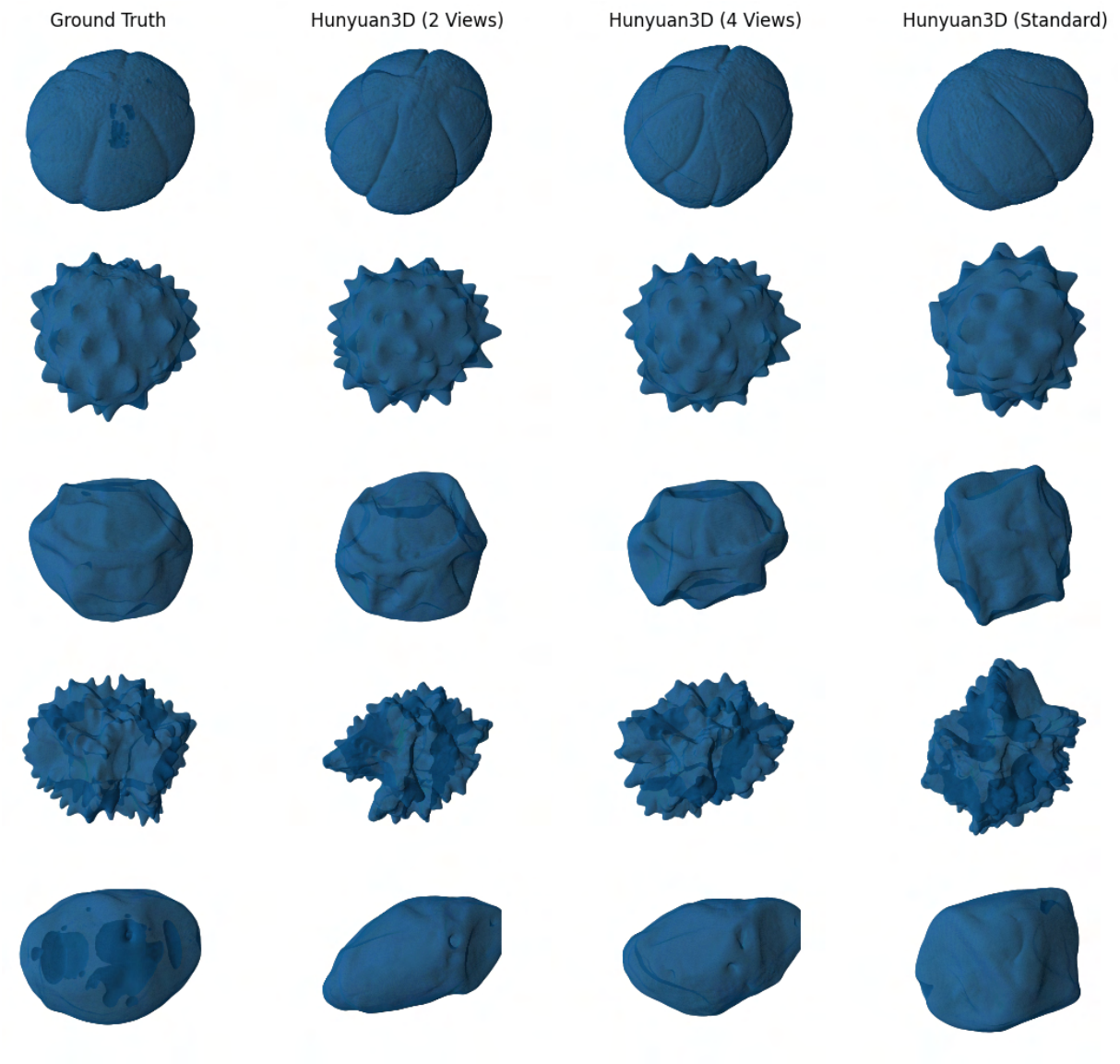


Figure 5.3: Hunyuan3D-2 at 50 Inference Steps

Insights

Hunyuan3D-2 is optimized for perceptual mesh quality rather than volumetric accuracy. In our experiments, although the canonical setup uses three conditioning views, the *two-view* variant achieved the highest fidelity, with *four views* a close second.

Qualitatively, Hunyuan3D-2 continues to excel with round, spherical, or ellipsoidal grains, but may still hallucinate fine spiky details under extreme occlusion. This behavior likely reflects its broad pretraining on graphics-oriented collections (e.g., Objaverse, ShapeNet, 3DScanRep, ModelNet40, Thingi10K). Tencent's announced refinement pipeline could even further shift its focus toward shape-accurate reconstructions, which would benefit highly irregular pollen.

1. **Hunyuan3D-two-views (front, left)**: Achieves the lowest Chamfer distance (0.0432 ± 0.0126) and highest IoU ($79.05 \pm 9.58\%$). With only two orthogonal inputs, the extra inference iterations compensate for the limited coverage, yielding sharp spines and faithful ornamentation, even on moderately complex grains.
2. **Hunyuan3D-four-views (front, left, right, back)**: Ranks second with Chamfer 0.0443 ± 0.0129 and IoU $77.89 \pm 9.90\%$. The added back view improves the recall of occluded surfaces, although slight flow-matching artefacts may appear where conflicting gradients arise.
3. **Hunyuan3D-three-views (front, left, right)**: Now third, recording Chamfer 0.0477 ± 0.0148 and IoU $76.02 \pm 10.38\%$. Although it still recovers most protrusions faithfully, the increased diffusion steps amplify the minor inconsistencies between the three inputs.

Taken together, these findings demonstrate that, with sufficient inference iterations, Hunyuan3D-2 can produce its highest-resolution pollen meshes even from just two views, while additional views yield diminishing returns or slight degradations for highly complex specimens.

A significant drawback, clearly visible in the plots, is that Hunyuan3D-2 often hallucinates structures that are not present in the input images, which is a side effect of its diffusion-based reconstruction.

5.4 Zero-Shot Modality Transfer on Holographic Pollen Images

The goal in this experiment is to assess to what degree our models can transfer the reconstruction capability learned from our synthetic 3D Pollen Library dataset to real-world holographic pollen scans. For this experiment we use seven hand-picked species from pollen grain scans of the Swisens Poleno [2] monitoring system. In a simple preprocessing step, we cut out the pollen grains on the holographic scans, resulting in images where only the grain itself was visible on a white background (The method of how we preprocess these samples can be found in Appendix A.9). Thus, we removed interference fringes in order to move closer to our synthetic dataset, which shows a white background and gray mesh faces across all samples. The closest representation that is available to such systems, as we have already established in Section 3.2, is the Visual Hull. In the Visual Hull Section 3.2 we learned that the Visual Hull reconstructs the largest volume that reproduces all silhouettes. Thus, we can exploit it as a somewhat plausible ground truth bounding shape to obtain metrics describing how well our estimators fit this "worst-case" reconstruction. We therefore calculate our metrics between the Visual Hull "ground truth" and the predictions of Pixel2Mesh++, PixelNeRF and Pix2Vox. It is important to notice that in this case, the metrics do not directly describe the reconstruction capability for holographic scans, but rather how similar our reconstructions are compared to the basic reconstruction of the Visual Hull, which is a widely adopted and cheap 3D representation that can be obtained in such systems without having to train a model.

Experiment Settings

1. **Visual Hull**: The Visual Hull reconstruction was used as a baseline.
2. **Pixel2Mesh++**: We used the fine-tuned Pixel2Mesh++ model trained on our augmented holo-pollen dataset and selected the best-performing checkpoint for evaluation (see fine-tuning experiment in Table 5.2).
3. **PixelNeRF**: We used the model trained from scratch without the ShapeNet weights, but with the augmented data and an ImageNet-pretrained encoder weights, to perform zero-shot inference on our holo-pollen images. Since this setting yielded the best metrics see: 5.3 5.3.
4. **Pix2Vox**: For Pix2Vox, we utilized the model trained from scratch on the augmented dataset, as this setting yielded the lowest Chamfer distance and highest IoU.

Quantitative Results

Model	Chamfer	F-Score			IoU
		$\tau_1 = 1\%$	$\tau_2 = 2.5\%$	$\tau_3 = 5\%$	
Pixel2Mesh++	$\mathbf{0.170 \pm 0.046}$	$\mathbf{8.3 \pm 2.6}$	$\mathbf{34.6 \pm 8.1}$	$\mathbf{51.3 \pm 9.6}$	$\mathbf{34.7 \pm 9.1}$
PixelNeRF	0.200 ± 0.064	7.9 ± 2.9	27.8 ± 10.8	42.5 ± 14.9	28.3 ± 18.0
Pix2Vox	0.236 ± 0.066	6.7 ± 2.1	22.1 ± 8.7	34.8 ± 12.8	16.5 ± 13.9

Table 5.5: Evaluation metrics of all compared models on the new augmented holo-pollen dataset. Results are mean ± standard deviation. The global best metrics are bold.

Qualitative Results

In subsequent experiments, the holographic input images were upsampled to a standardised resolution. The applied scaling factor was limited to a maximum of 1.7× to preserve the fidelity and prevent artefacts introduced by excessive interpolation.

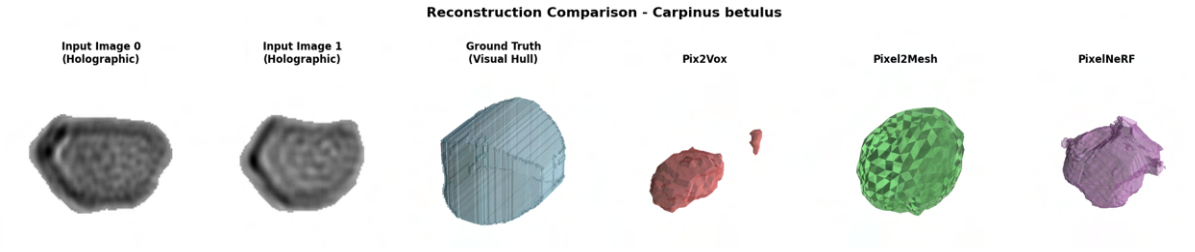


Figure 5.4: Holographic reconstruction of *Carpinus betulus*. Pix2Vox produces plausible structures; Pixel2Mesh overestimates spherical features; PixelNeRF, conversely, underestimates spherical shapes.

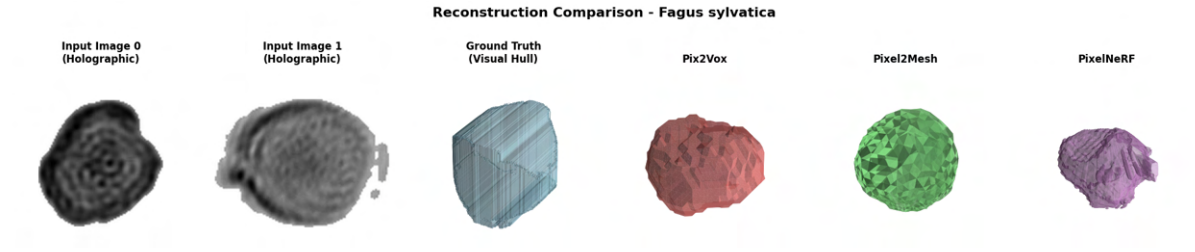


Figure 5.5: Holographic reconstruction of *Fagus sylvatica*. All models exhibit structural issues: the meshes fail to capture the flat, bean-like profile suggested by the hologram.

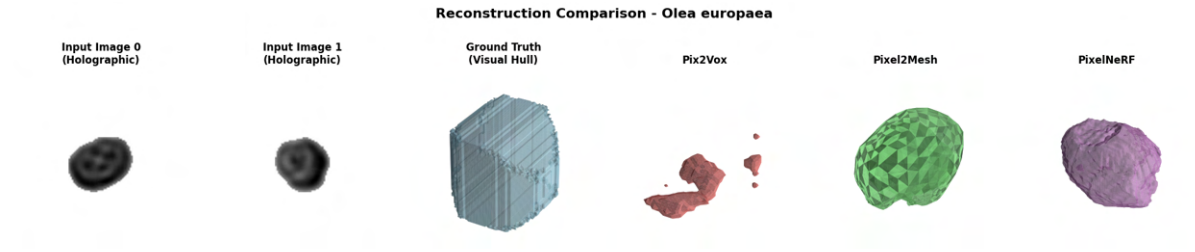


Figure 5.6: Holographic reconstruction of *Olea europaea*. At lower resolution settings, Pix2Vox encounters a bottleneck that leads to overly coarse meshes.

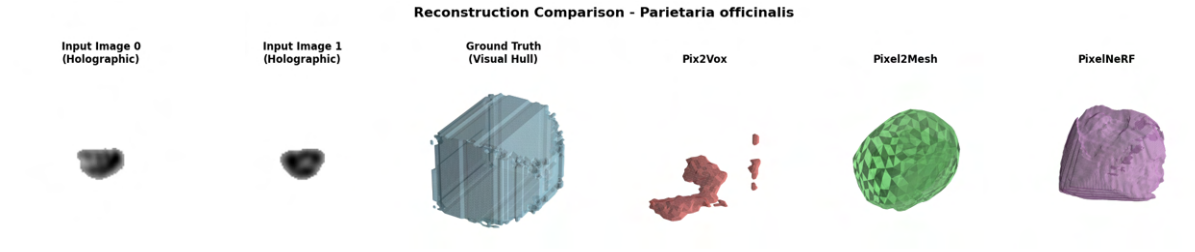


Figure 5.7: Holographic reconstruction of *Parietaria officinalis*.

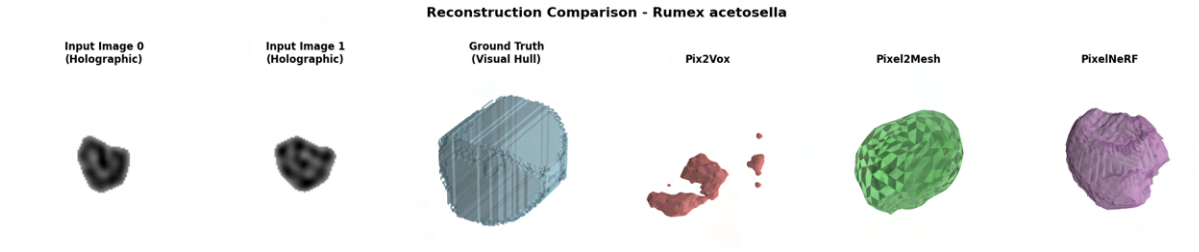


Figure 5.8: Holographic reconstruction of *Rumex acetosella*.

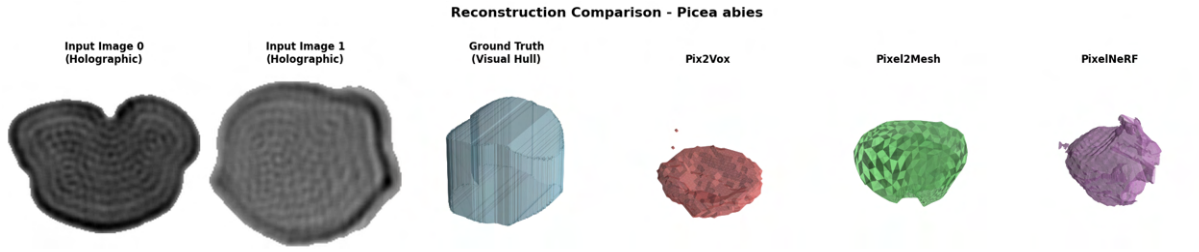


Figure 5.9: Holographic reconstruction of *Picea abies*. Pix2Vox yields overly flat meshes; Pixel2Mesh produces excessively spherical shapes; PixelNeRF is somewhat noisy but better preserves the overall structure.

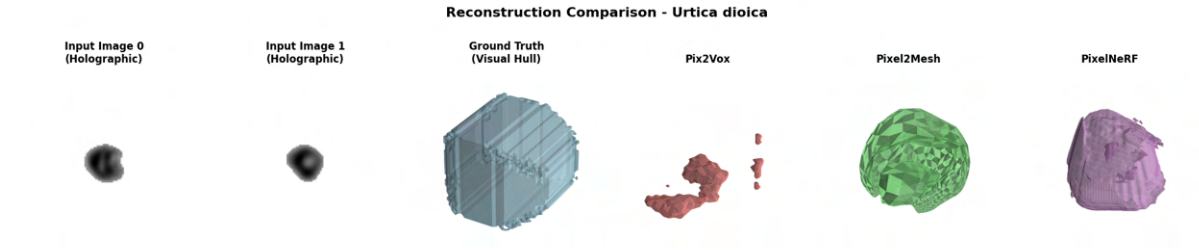


Figure 5.10: Holographic reconstruction of *Urtica dioica*.

Insights

Overall, the reconstruction quality increases substantially with a higher *effective* input resolution: all three networks exhibit difficulties when processing low-resolution inputs but show marked improvement once the images are upscaled or resized. Since the Visual Hull used for evaluation is only an *approximate* geometric proxy, the absolute values in Table 5.5 should be interpreted as indicative rather than definitive ground truth.

The Visual Hull was chosen as an evaluation baseline because it provides the most geometrically consistent and reproducible reference available under our constraints. Furthermore, it is a deterministic and model-agnostic method that can be applied in any multi-view scenario without requiring training data or learned priors. Its simplicity, general applicability, and robustness make it a suitable benchmark against which learned reconstruction methods can be critically compared.

1. **Pix2Vox**:
   - Fragmented, “blobby” outputs when the inputs are small or noisy; often fails to reconstruct a single cohesive object.

- Even with larger inputs the meshes remain too flat, missing vertical features visible in the holograms.
- Scores the highest Chamfer Distance and lowest IoU, reflecting its lack of an implicit representation or shape prior.

2. **PixelNeRF**:
   - Captures bumps and edges that match the input views, but introduces noticeable noise, especially with pixelated crops.
   - Quantitatively trails Pixel2Mesh++ but outperforms Pix2Vox across all metrics.
3. **Pixel2Mesh++**:
   - Achieves the *lowest* Chamfer Distance and *highest* IoU in Table 5.5.
   - Produces reliable, nearly spherical reconstructions of pollen grains, though it can overshoot height/depth and appear over-rounded.
   - Benefits most from higher-resolution inputs; up-scaling and sharpening noticeably reduce surface wobble.

5.5 Reconstruction Limitations: Number of Pollen Views

Our fifth experiment concerns the number of input views. Within this experiment we aim to see whether reconstruction fidelity improves if we add more than just two orthogonal views during training and inference. We mainly considered the range of 1 to 6 input views. We take the first six of the preprocessed input views in the fixed rotations for every model as described in Section 4.4.4.

Experiment Settings

1. **Visual Hull**: We use the default settings for the Visual Hull estimator and run evaluation from 1 to 6 views.
2. **Pix2Vox**: We run training on the default settings laid out in Section 3.3. The model gets trained and evaluated from 1 to 6 input images.
3. **PixelNeRF**: The same experiment settings as for Pix2Vox apply here. We evaluate the model from 1 to 6 input images, however we do not retrain the model on a different number of views as PixelNeRF is already trained on 128 views. Additionally, we note that training samples are decoupled from inference samples in PixelNeRF.
4. **Pixel2Mesh++**: Unlike the other models in the Pixel2Mesh++ run, we only look at the input view range of 2-6 images in Pixel2Mesh++. This is due to the Cross-view Perceptual feature pooling that requires at least two images to pool from. Accommodating single-view reconstructions would have meant significant architectural changes, which is why we have decided against this path in the thesis.

Quantitative Results

Model	# Views	Chamfer	F-Score			IoU
			$\tau_1 = 1\%$	$\tau_2 = 2.5\%$	$\tau_3 = 5\%$	
Visual Hull	1	0.134 ± 0.026	9.1 ± 2.5	35.0 ± 6.4	54.7 ± 7.6	52.5 ± 9.3
	2	0.091 ± 0.025	10.9 ± 4.0	41.6 ± 10.2	66.4 ± 12.1	63.7 ± 9.1
	3	0.093 ± 0.025	11.0 ± 3.6	42.1 ± 10.0	66.2 ± 11.5	63.4 ± 9.5
	4	0.089 ± 0.023	11.3 ± 4.0	43.0 ± 10.4	67.6 ± 11.4	64.4 ± 8.7
	5	0.079 ± 0.034	13.1 ± 5.8	49.1 ± 15.6	73.7 ± 16.0	68.0 ± 12.9
	6	0.058 ± 0.023	18.3 ± 8.0	62.8 ± 18.9	84.6 ± 13.4	76.6 ± 10.2
Pix2Vox	1	0.062 ± 0.020	14.8 ± 4.8	57.7 ± 12.7	83.2 ± 11.5	72.4 ± 10.9
	2	0.060 ± 0.021	16.7 ± 6.1	60.3 ± 15.1	84.8 ± 12.1	71.4 ± 17.6
	3	0.055 ± 0.019	17.8 ± 6.9	63.6 ± 14.3	86.5 ± 11.0	73.9 ± 12.9
	4	0.057 ± 0.021	16.9 ± 7.0	61.5 ± 16.8	85.5 ± 13.0	76.6 ± 9.4
	5	0.056 ± 0.017	16.8 ± 5.4	61.2 ± 13.6	86.7 ± 10.1	75.5 ± 9.1
	6	0.055 ± 0.018	17.6 ± 6.2	63.9 ± 13.9	86.9 ± 11.5	74.7 ± 11.9
PixelNeRF	1	0.059 ± 0.012	17.2 ± 4.6	61.2 ± 7.5	84.2 ± 7.6	75.5 ± 11.9
	2	0.043 ± 0.013	26.0 ± 8.9	76.2 ± 12.3	$\mathbf{90.9 \pm 8.3}$	82.8 ± 6.5
	3	0.041 ± 0.016	28.6 ± 11.5	80.0 ± 15.3	91.2 ± 9.5	83.7 ± 7.6
	4	0.040 ± 0.014	28.9 ± 13.1	78.8 ± 16.2	91.8 ± 7.9	82.0 ± 8.9
	5	0.044 ± 0.018	28.9 ± 14.2	77.5 ± 17.1	89.8 ± 9.8	82.7 ± 11.7
	6	$\mathbf{0.040 \pm 0.012}$	$\mathbf{32.7 \pm 12.8}$	$\mathbf{80.9 \pm 13.7}$	90.7 ± 7.6	$\mathbf{85.8 \pm 7.8}$
Pixel2Mesh++	2	0.052 ± 0.023	21.3 ± 9.0	69.1 ± 15.6	88.5 ± 12.5	40.6 ± 23.4
	3	0.056 ± 0.024	18.8 ± 8.7	64.4 ± 15.5	86.8 ± 13.3	70.1 ± 16.2
	4	0.058 ± 0.026	18.9 ± 9.0	64.2 ± 16.7	84.8 ± 14.2	73.7 ± 13.0
	5	0.054 ± 0.028	19.6 ± 6.8	67.4 ± 15.0	87.5 ± 13.3	40.5 ± 24.5
	6	0.052 ± 0.022	20.0 ± 8.9	68.1 ± 15.6	88.1 ± 12.2	41.0 ± 23.0

Table 5.6: Evaluation metrics of models under a different number of views used for reconstruction. Plots of these quantitative results showing improvement curves can be found in Appendix A.3. The best scores within each model are underlined, the global best metric across models is bold.

Qualitative Results

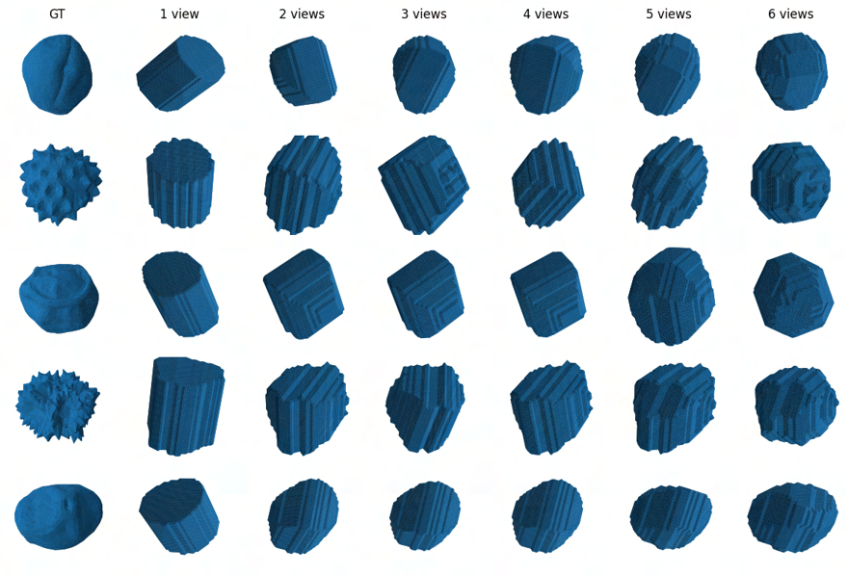


Figure 5.11: Visual Hull. Qualitative plot showing our five samples evaluated with 1 through 6 input views.

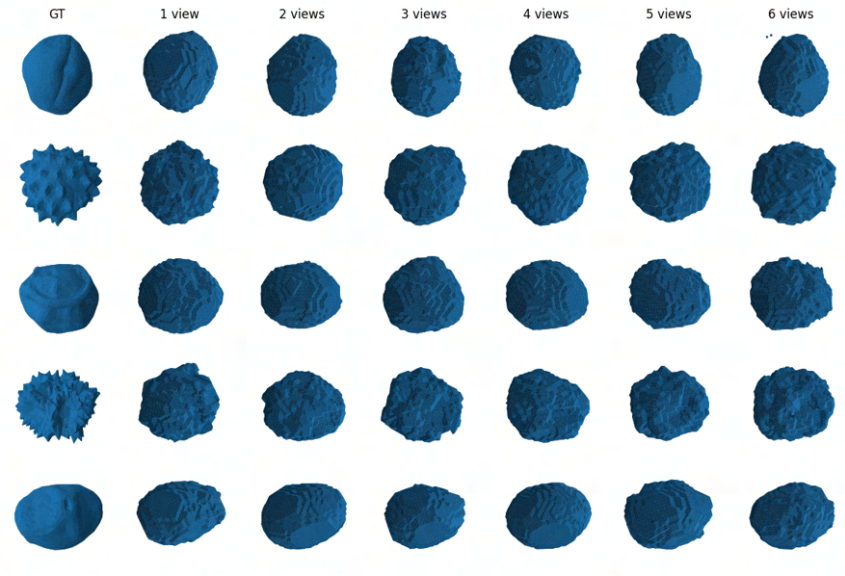


Figure 5.12: Pix2Vox. Qualitative plot showing our five samples evaluated with 1 through 6 input views.

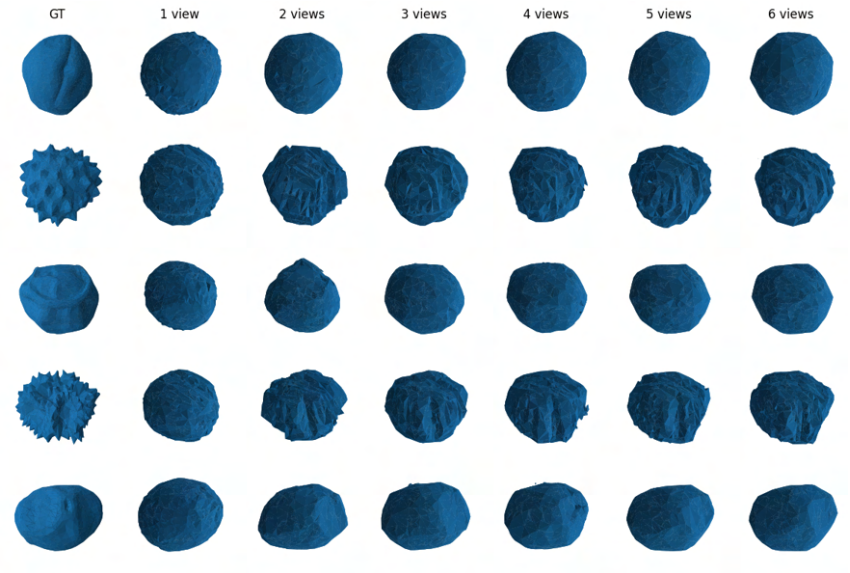


Figure 5.13: PixelNeRF. Qualitative plot showing our five samples evaluated with 1 through 6 input views.

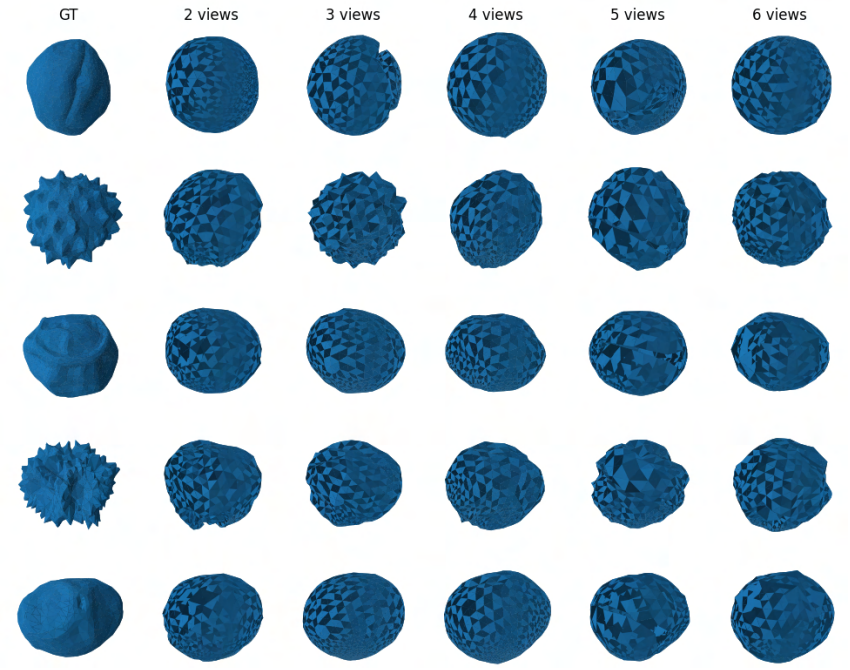


Figure 5.14: Pixel2Mesh++. Qualitative plot showing our five samples evaluated with 2 through 6 input views.

Insights

1. **Visual Hull**: In the metrics we can see quite a clear improvement when more views are added. Between 2 and 3 views the improvement is less apparent, both quantitatively and qualitatively. This is most likely an effect of the third view angle that is added. The first view sits at 0° and the third at 180°. Thus, they sit opposite of each other. This means that if the mesh is somewhat symmetric, Visual Hull will not carve out much more information that could be beneficial for reconstruction.

2. **Pix2Vox**: The metrics are fluctuating more and show less a clear improvement with more views. The starkest improvement can be seen between a single-view and two views. Afterwards, the improvements are less clear. Across all metrics, the Chamfer distance shows the clearest improvements. Qualitatively, the Pix2Vox model roughly shows the same observation. Starting from 2 views, the reconstruction becomes plausible in terms of general shape, however the model still struggles with more fine details, even when 6 images are available to learn from. We suppose the less clear improvement stems from the fact that Pix2Vox just learns a binary classification task from input images. The model does not learn how views relate to each other in terms of angle, as mentioned in Section 3.3.
3. **PixelNeRF**: Quantitatively, the PixelNeRF model performs the best in all view categories. Similarly to the Visual Hull runs, this model shows clear improvement when more views are added for reconstruction. Interestingly, between 4 and 5 views, the metrics do not improve across metrics. At 6 views the model performs the best. We therefore assume that the stagnation (or slight dip) between 4 and 5 views can be traced back to the statistical insignificance of our results, due to the low number of samples with high variability, analogous to the high standard deviation. Qualitatively, the model captures general shape somewhat well, similar to the qualitative performance of Pix2Vox. The more difficult pollen samples however seem to be reconstructed with a more correctly textured surface. The improvement is less apparent in the qualitative samples. The most improvement arguably happens between 2 and 3 views. In the 2-view run, some samples show a few structural inconsistencies (for example the pointy top in the third row, or the overly jagged second row sample). The model looks to fix those with 3 or more views.
4. **Pixel2Mesh++**: Similar to the PixelNeRF results, and in terms of metrics, this model seems to benefit from having more views added. Parallel to the inconsistencies mentioned in the PixelNeRF run, we can also see some fluctuations in performance here, likely to stem from the small, high-variance evaluation set. The Pixel2Mesh++ IoU metric also shows that more views do not consistently produce watertight shapes. The IoU is still in a comparable regime when looking at Pixel2Mesh++ in the other experiments. Qualitatively, it remains hard to see a clear reconstructed texture. The model is able to catch the general shape, close but not as good as the qualitative results of Pix2Vox and PixelNeRF. Though, it is much clearer that the experiment runs produced quite differently shaped reconstructions.

Over all, the number of views ablation produced quite differing results among the models. The Visual Hull reconstructions underline the importance of having views at multiple different angles that do not lie directly in opposition to other views. The Pix2Vox model arguably learns the most when jumping from just a single-view to two views. Further views do not add much more noticeable improvements to the qualitative evaluation samples. This is likely due to the model's nature that starts to saturate its capacity and the merger does not gain more valuable "coarse shape" information from more views. The model succeeds at recovering the general shape with just two images, especially if the pollen grain is simple in its topology. PixelNeRF is very much comparable to Pix2Vox's performance. It gains the most insight when jumping from single-view-reconstruction to dual-view-reconstruction. The general shape looks visibly correct in the reconstructions that use more than a single-view. It however improves the reconstructions further when it comes to texture when adding more views around the pollen grains. Lastly, the Pixel2Mesh++ model shows less optimal improvements. There seems to be no clear improvement when adding more views. The reconstructed shapes somewhat take up the general shape of the ground truth, however the general shape fidelity is not as good as within Pix2Vox and PixelNeRF. The Visual Hull reconstruction metrically benefits the most from more views. At just a single view it reaches an average Chamfer Distance of 0.134 while it reduces by $\approx 56.72\%$ with 6 views, hitting a Chamfer Distance of 0.058.

5.6 Explicit Pollen Priors with Pixel2Mesh++

To answer RQ2 ("Do explicit priors of pollen shapes improve the quality of 3D reconstructions from 2D images?"), we take a closer look at the template mesh of Pixel2Mesh++. As we discussed in Section 3.4, the Pixel2Mesh++ model deforms a template mesh [9], [22]. In this experiment we try to exploit this template mesh by replacing it with three different "explicit priors":

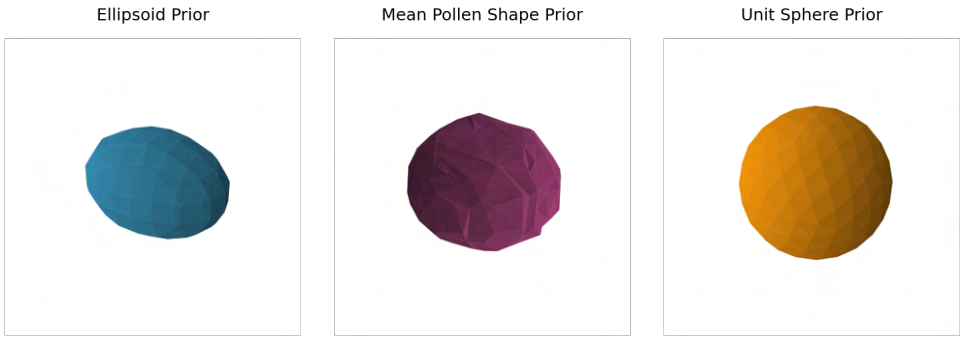


Figure 5.15: Our Priors. Default: ellipsoid (original template by Wang, Zhang, Li, *et al.* [22]), mean shape: mean overall shape across the pollen meshes in the training partition, unit sphere: a perfect canonical sphere

We directly inherit the ellipsoid from the original Pixel2Mesh publication by Wang, Zhang, Li, *et al.* [22]. Since the model was trained on this 156-vertex template, we made sure that our newly added Mean Pollen Shape and Unit Sphere Priors follow the same graph structure of 156-vertices. The construction of such meshes is non-trivial. We discuss the method of obtaining these two additional priors in Appendix A.7.

Quantitative Results

Model	Prior	Chamfer	F-Score			IoU
			$\tau_1 = 1\%$	$\tau_2 = 2.5\%$	$\tau_3 = 5\%$	
Pixel2Mesh++	Ellipsoid	$\mathbf{0.051 \pm 0.018}$	20.2 ± 8.1	67.7 ± 11.6	88.0 ± 9.6	43.1 ± 22.6
	Mean Pollen	0.054 ± 0.018	19.3 ± 4.6	66.6 ± 10.6	87.5 ± 9.5	$\mathbf{75.9 \pm 10.0}$
	Spherical	0.052 ± 0.023	$\mathbf{21.3 \pm 9.0}$	$\mathbf{69.1 \pm 15.6}$	$\mathbf{88.5 \pm 12.5}$	40.6 ± 23.4

Table 5.7: Evaluation metrics of models compared by base dataset training and augmented dataset training. The global best metrics are bold.

Qualitative Results

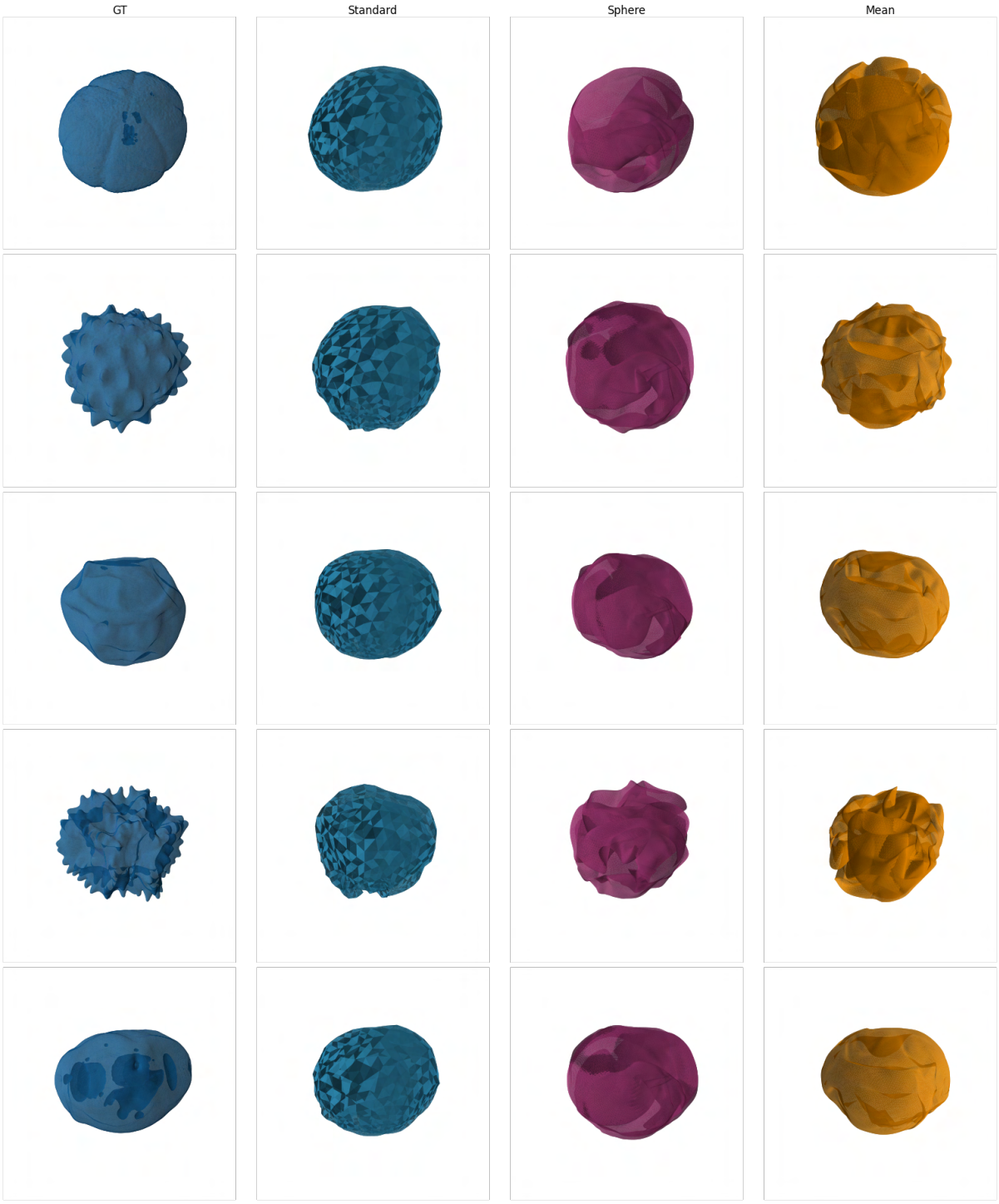


Figure 5.16: Qualitative results of our different prior strategies on five of our evaluation samples.

Insights

1. **Ellipsoid Prior**: The default Ellipsoid Prior yields the lowest Chamfer Distance across all setups, however it does not significantly improve compared to the other priors. Neither do the other metrics show a significantly better or worse result. The lower chamfer distance likely stems from the model initializing with pretrained weights that may help the model

transfer to pollen better. Qualitatively, the run with the ellipsoid prior, as seen in previous experiments, roughly captures the general shape of the ground truths.

2. **Mean Pollen Prior**: Similar to the Ellipsoid Prior, the Mean Pollen Prior does not show any significantly better or worse results across our metrics except for the IoU. The IoU shows a significant advancement, including a lower standard deviation. This improvement shows that the recovered volumes are likely more watertight and produce a more coherent pollen structure. In the qualitative view, the Mean Pollen Prior shows clearer textures for the evaluation samples with spiky surfaces. For the first evaluation sample, it however struggles to capture the general shape, and rather recovers a too large shape.
3. **Spherical Prior**: The run with the Spherical Prior performs almost identically to the Default Ellipsoid Prior across all metrics. The selected evaluation samples also show comparable results to the Ellipsoid Prior run. This indicates that the Pixel2Mesh++ model does not significantly improve or worsen if the template becomes a symmetrical ellipsoid (unit sphere).

Apart from the significant improvement of the IoU, when training the model with a mean shape prior, this experiment does not show to benefit from one of the added priors. The Mean Shape Prior increased the IoU significantly, showing that it seems to largely fix the issue of the meshes becoming non-watertight with larger apertures. Overall, our prior templates qualitatively produce clearer reconstructions as the "checkerboard" texture can be avoided. This is due to the fact that we have all faces available to render a resulting mesh. Unfortunately, the renderer we use in our qualitative experiments struggles with the missing face information that is not published by Wen, Zhang, Li, *et al.* [9] and produces a "checkered" pattern.

6 Discussion & Limitations

6.1 RQ 1: Best-suited reconstruction technique

Under the two-view constraint, the methods we evaluated separate cleanly into three behaviors: (i) purely geometric baselines that under-fit concavities (Visual Hull), (ii) learned predictors whose output representation limits either resolution or watertightness (Pix2Vox, Pixel2Mesh++), and (iii) implicit field or large-model approaches that leverage strong priors to interpolate the unseen views (PixelNeRF, Hunyuan3D-2). Quantitatively and qualitatively, the encoder-conditioned implicit model (PixelNeRF) offers the best overall trade-off between accuracy, stability, and resource use for two orthogonal inputs on our pollen benchmark.

PixelNeRF
With two inputs, PixelNeRF attains the strongest balanced score profile: CD = 0.043, F-Score@τ_3 = 90.9%, and IoU = 82.8%. Beyond two views, PixelNeRF scales gracefully, but the two-view metrics already set the top line among trained methods. PixelNeRF additionally also performed comparable regardless of training it on the very sparse base dataset or the augmented dataset. Thus, it is a well-fit method for the domain of pollen grain reconstruction, where 3D data is not readily available.

Pix2Vox
Pix2Vox recovers the gross volume from two views (CD = 0.060, F-Score@τ_3 = 84.8%, IoU = 71.4%) but systematically blurs spikes and grooves at the 32^3 output resolution. It benefits modestly from more views and from our augmentation, yet remains behind PixelNeRF on all metrics in the two-view setting due to discretization and limited cross-view reasoning.

Pixel2Mesh++ (mesh deformation)
At two views, Pixel2Mesh++ reaches CD = 0.052 and F-Score@τ_3 = 88.5%. This is in a competitive surface proximity, but its IoU collapses ($40.6\% \pm 23.4\%$) because many predictions are not watertight, a known fragility of the released implementation [9], [22]. Additional views do not consistently help; metrics oscillate and mostly fail to surpass the two-view baseline. The results show the limited multi-view fusion in this model.

Hunyuan3D-2
When run at a high-quality setting (50 flow-matching steps, dense SDF decode), Hunyuan3D-2 in a two-view configuration nearly ties PixelNeRF on surface metrics (CD = 0.0432, F-Score@τ_3 = 91.15%) but lags on IoU (79.05%). In our ablations, supplying more than two inputs produced no consistent gains and sometimes small regressions, likely due to pre-training biases toward single/few-view conditioning. Crucially, the method is also far more resource-intensive than PixelNeRF to reach these scores. The method however does show much better capabilities in the qualitative recovery of textures, as this is also one of the main use-cases Hunyuan3D-2 was trained for [14].

Given two orthogonal images, PixelNeRF is presently the best-suited technique on our dataset: it delivers the strongest overall accuracy (top or near-top Chamfer/F-Score with the highest IoU among learned methods), and behaves robustly across shapes, making it practical for the under-parametrized task we face. Hunyuan3D-2 may be a compelling alternative when heavy compute is acceptable: in zero-shot it follows PixelNeRF closely on surface fidelity and can recover the fine pollen grain textures, but it is less reliable on volumetric occupancy and substantially more demanding to run. Pix2Vox is a useful coarse baseline and captures the general pollen grain shape in most cases. Pixel2Mesh++ shows acceptable surface proximity but is undermined by watertightness failures. This has driven the IoU down consistently throughout our experiments.

6.2 RQ 2: Explicit priors of pollen shapes

We probed explicit shape priors only in Pixel2Mesh++, whose pipeline deforms a fixed 156-vertex template (coarse Pixel2Mesh Multi-View Deformation Network). Because the architecture assumes that exact vertex budget and a single global template for all samples, every prior must share the same 156-vertex graph; we therefore tested three templates: the original ellipsoid, a mean pollen shape, and a unit sphere.

Quantitatively (two-view setting), swapping the template left surface-accuracy metrics essentially unchanged: Chamfer hovered at $\approx$ 0.051–0.054 and F-Score@τ_3 = 88% across priors. The only clear shift was IoU, which jumped from 43.1% (ellipsoid) to 75.9% (mean prior), with reduced variance; the unit sphere behaved like the ellipsoid (IoU 40.6%).

No surface gains while IoU improves
IoU in our setup is highly sensitive to watertightness due to calculating the metric via voxels. Pixel2Mesh++ frequently produced open meshes / missing faces (including the "checkerboard" surface as seen on qualitative plots) when driven from the default prior. When such a reconstructed mesh has apertures, the voxelization process for such a mesh results in voxels just appearing at the surface, but not filling up the inside of the mesh. Adopting a mean prior reduced these failures, hence better volumetric occupancy, even though point-wise surface proximity (Chamfer/F-Score) did not improve. In short, the mean prior regularizes volume, not fine geometry. Qualitatively, it sometimes clarified spiky textures but could over-inflate the overall shape on specific cases.

Architectural sensitivity to the prior
Pixel2Mesh++ is implicitly tuned for a its ellipsoid template; replacing it with a more anisotropic average pollen can induce implausible deformations unless other parts of the system (losses, Laplacian/edge regularisers, upsampling schedule) are re-tuned. Our experiments and discussion notes reflect this fragility: prior swaps alone do not yield systematic surface improvements and can re-introduce face/connectivity issues without code-level fixes.

Limitations to keep in mind
(i) A single global template per training run (imposed by the 156-vertex requirement) prevents species-specific priors; (ii) we did not re-optimise the deformation hierarchy per prior; (iii) renderer artefacts magnified the visual impact of missing faces in some comparisons. Together, these factors bias the outcome toward volumetric stability gains rather than genuine surface fidelity gains.

To answer the research question, explicit pollen priors do not improve surface accuracy (Chamfer and F-Score) for the Pixel2Mesh++ model. A mean-shape prior does however substantially improve IoU by promoting watertight, coherent volumes. To turn explicit priors into consistent accuracy gains, we would likely need to implement architecture-level adaptations (regularization, deformation schedules, perhaps per-class templates) rather than a plug-in template swap.

6.3 RQ 3: Number of views

Two orthogonal views are enough to recover plausible pollen geometry with learned implicit priors (PixelNeRF) and even to rival large diffusion priors (Hunyuan3D-2), but they cap volumetric completeness and fine morphology. Adding views generally helps, but the gain is strongly model-dependent and quickly saturates beyond two inputs for our best performers.

Model-specific improvements

- **PixelNeRF**. Going from 1 to 2 views yields the biggest jump (CD = 0.059 to CD = 0.043, IoU = 75.5% to IoU = 82.8%). Beyond that, improvements are modest and non-monotonic; 6 views is best but only slightly better than 2 (CD = 0.040; IoU = 85.8%). Thus, most of PixelNeRF's benefit is already realized at two orthogonal images.
- **Visual Hull**. Two views under-carve concavities by design (IoU = 63.7%), and more coverage helps substantially; by 6 views the hull tightens markedly (CD = 0.058, IoU = 76.6%). Note the small gain from 2 to 3 views because the third view at 180° adds little on near-symmetric grains.
- **Pix2Vox**. The clearest step is 1 to 2 views (better Chamfer, higher F-Score), with smaller, fluctuating gains thereafter. It shows order-invariant fusion learns a "coarse shape" but cannot recover precise cross-view geometry and smaller texture details.
- **Pixel2Mesh++**. Extra views seem to not reliably help. The Chamfer Distance hovers around 0.052 and 0.058, while IoU oscillates (e.g., 40.6% at 2 views to 73.7% at 4 and 41.0% at 6), reflecting unstable watertightness rather than systematic multi-view fusion.
- **Hunyuan3D-2**. Counter-intuitively, 2 views are best in our runs (CD = 0.0432, IoU = 79.05%), with slight degradations at 3 to 4 views. This is likely a pretraining bias toward single/few-view conditioning followed in Zhao, Lai, Lin, *et al.* [14].

Why two views are a hard ceiling

Two orthogonal silhouettes simply cannot reveal invisible concavities; analytic methods over-fill them and learned ones must hallucinate from priors. This explains both the strong 2 to 6-view gains for Visual Hull and the quick saturation for PixelNeRF once priors have "filled in" the unseen structure.

Generally, the biggest leap is 1 to 2 views (especially for PixelNeRF), with diminishing returns afterwards. Additionally, view placement matters: adding an opposite (180°) view helps less than adding a novel angle, especially when pollen grains are symmetrical.

Conclusively, with only two orthogonal images, reconstruction quality is constrained but not impossible. PixelNeRF already achieves near-final accuracy at two views, with only $\approx 3\%$ IoU points left on the table by six views. Visual Hull, by contrast, remains significantly under-carved at two views and needs many more to approach learned methods. Template-based and diffusion models show inconsistent gains unless specifically optimized for multi-view fusion. In practice, if data collection is limited to two captures, choose an implicit prior (PixelNeRF) and expect good global shape with some loss of concavity fidelity and fine ornamentation.

6.4 RQ 4: Holographic images

We tested zero-shot modality transfer from our synthetic 3D Pollen Library to seven holographic crops from a Swisens Poleno device. To narrow the domain gap, holograms were tightly cropped and the interference fringes removed, yielding white-background inputs closer to our training renders. Because no true 3D ground truth exists for these scans, we compared each prediction against a Visual Hull built from the same two views. Thus, the reported metrics measure similarity to a conservative proxy, not real-world accuracy.

What transferred and what didn't

Across models trained on synthetic data, transfer was possible but limited:

- Pixel2Mesh++ scored best against the hull proxy (CD = 0.170 ± 0.046, IoU = $34.7\% \pm 9.1\%$), tending to produce smooth, near-spherical meshes that align volumetrically with the convex bias of Visual Hull. Upscaling the holograms ($\leq 1.7\times$) helped most here, reducing surface wobble. Qualitatively, however, it sometimes over-rounded shapes relative to the hologram evidence.
- PixelNeRF transferred plausible edge/relief cues from the holograms (e.g., flattening where the images suggest it), but predictions were noisier and scored lower vs. the hull (CD = 0.200 ± 0.064, IoU = $28.3\% \pm 18.0\%$). Still, it consistently outperformed Pix2Vox on all metrics.
- Pix2Vox degraded the most on holograms (CD = 0.236 ± 0.066, IoU = $16.5\% \pm 13.9\%$), often resulting in flat or fragmented shapes. This is however unsurprising given the lack of a strong learned prior and its coarse 32^3 output.

Qualitatively across the evaluation sample species, Pixel2Mesh++ tended to "oversphere," Pix2Vox stayed too flat/coarse, and PixelNeRF better respected silhouette-driven flattening but showed roughness when optimization was not specialized for the modality.

Limitations of this experiment

Without true 3D ground truth for holograms, we cannot certify absolute accuracy; Visual Hull was a practical stand-in, not a target morphology. Moreover, our hologram handling removed fringes (and thus some depth cues), so results reflect domain harmonization rather than native holographic reconstruction. Finally, we did not provide quantitative hologram results for Hunyuan3D-2; we note that this would require ground truth or a stronger evaluation protocol since Hunyuan3D resorted to constructing other objects it learned from the Objaverse [44] dataset in preliminary tests.

Conclusively, with light preprocessing and no hologram-specific training, some of the synthetic-trained methods do produce plausible reconstructions on real holographic inputs: Pixel2Mesh++ aligns best with a Visual-Hull reference, while PixelNeRF better tracks morphology hinted by the images but pays a penalty in hull-centric metrics. However, the adaptation remains constrained by the lack of hologram ground truth and by domain cues lost in the fringe removal. In short, synthetic-to-real transfer is feasible but not yet dependable for fine morphology.

6.5 Limitations

Dataset sparsity and heterogeneity
Our benchmark comprises just 32 evaluation meshes, whereby all of them are of unique species. This extreme species-level sparsity, combined with large morphological variability, makes generalization and significance testing challenging. We intentionally avoided species leakage across splits and relied on augmentation to mitigate the "one-shot per species" regime, but the underlying variability remains high.

This also has a direct effect on our quantitative results. Across experiments, standard deviations are large, many apparent differences are not statistically robust. As noted in the results section, several comparisons are "not clearly significant" due to this dispersion. Our experiment results therefore need to be examined with caution.

Modality transfer constraints
For holographic inputs, we lacked 3D ground truth and thus compared predictions against a Visual Hull proxy built from the same two views. Reported scores therefore reflect similarity to a conservative hull, not absolute real-world accuracy, and the sample size was small (seven crops)

Model coverage and compute
Several recent diffusion-based systems (e.g., InstantMesh [17], SparseFusion [19], Zero-1-to-3 [18]) exceeded our memory budget and were excluded. This limits direct comparability to certain state-of-the-art families and may bias the landscape toward approaches that fit within our resources.

7 Future Work

7.1 Improving Existing Models

Each of the models evaluated in this thesis has clear areas for improvements:

- **Pixel2Mesh++**: The most significant limitation of this model is its failure to produce watertight meshes, often resulting in a "checkered" artifact of missing faces. Future work should focus on incorporating a loss term that explicitly penalizes non-watertight geometry. Additionally, the rendering issues that cause these visual artifacts need to be addressed at a more fundamental level, potentially by modifying the rendering engine or the mesh generation process itself.
- **PixelNeRF**: While PixelNeRF performs well, its reconstructions can lack fine-grained detail, appearing overly smooth. This could be addressed by replacing the simple MLP architecture with more advanced scene representations, such as tri-planes or compact grid-based structures, which have shown success in other neural rendering tasks. Furthermore, integrating more powerful feature extractors, such as Vision Transformers (ViT) or pretrained models like DINO [34] or CLIP [45], could provide better spatial reasoning and semantic guidance, leading to more accurate reconstructions, especially in occluded regions.
- **Pix2Vox**: The performance of Pix2Vox could be improved by incorporating a more sophisticated fusion module that can better leverage the information from multiple views. Additionally, exploring a higher-resolution voxel grid, perhaps with an octree-based representation to manage memory, could allow for the reconstruction of finer details.

7.2 Exploration of Novel Models

This thesis focuses extensively on the concepts of NeRF and deformation. However, it is important to note that there exist alternative methods of reconstructing meshes with only sparse two-dimensional information. One such method is the Instant Mesh or Hunyuan3D-2, which employs diffusion to generate novel views and sample them into a three-dimensional mesh. However, the field of Gaussian Splatting has also experienced significant success in recent years. Gaussian Splatting has been shown to be faster and more efficient than NeRF, while its methodology for generating 3D volumes is comparable. However, it is evident that Gaussian Splatting is susceptible to the same vulnerabilities as PixelNeRF. The experiment demonstrated in Vanilla NeRF revealed that a minimum of 50 images are required. However, it should be noted that a paper and implementation of Gaussian Splatting, such as PixelNeRF, have been developed. This implementation is known as SplatterImage. Whilst PixelNeRF is more well-known, the two utilize the same input structure. It is evident that Gaussian Splatting consistently yields superior outcomes in comparison to vanilla NeRF, thus rendering it a suitable candidate for experimental investigation. This is particularly pertinent in the context of sparse reconstruction tasks involving only two orthogonal images, where gains could be expected.

Hunyuan3D 2.1 Training Pipeline Release

On **13 June 2025** Tencent's Hunyuan team open-sourced the entire training pipeline, model weights, and data loaders for `Hunyuan3D-2.1`[2]. The new checkpoint scales the shape branch to **3.1 B parameters** (versus 1.1 B in v2.0) and adds a physically-based texture generator, both of which are released under an MIT-style licence. The repository lists practical GPU footprints: *shape-only* inference fits in $\approx$ 10 GB VRAM, the PBR texture module needs $\approx$ 21 GB, and running both stages jointly requires $\approx$ 29 GB on a single card.

[2] `https://github.com/Tencent-Hunyuan/Hunyuan3D-2.1`

Although these figures are manageable for deployment, **fine-tuning** the 3.1 B–parameter DiT still calls for multi-GPU setups (80–120 GB aggregate VRAM, according to the maintainers). Integrating a pollen-specific dataset would therefore be an ambitious, but attractive-follow-up project for a future endeavour. The ability to adapt the latent prior to our domain could sharpen the reconstruction of exine ornamentation and aperture morphology, areas where generic pre-training occasionally falters. We leave this endeavour to future work, as our current study focuses on evaluating the frozen v2.0 weights under strict hardware limits (16 GB VRAM).

7.3 Holographic Images

The following subsection is concerned with the subject of holographic images. At the time of writing the thesis, the dataset for the holographic images had not yet been prepared. It should be noted that no ground truths were available for the purposes of testing or fine-tuning. A subsequent step would be to utilize the three-dimensional spatial information of the holographic traces for the training and improvement of our models. It is anticipated that, with the implementation of a suitable training framework, the holographic images will undergo a process of refinement in this setting. The 3D dataset under consideration is characterized by a significantly higher resolution, which results in the acquisition of a greater amount of information regarding the underlying 3D structure of the grain. Consequently, a trade-off between quality and reconstruction mesh is evident. However, it is hypothesized that this trade-off can be mitigated through the utilization of spatial information derived from a holographic image, see ripples 7.1. It is evident that the utilization of these informational elements would necessitate a degree of adaptation on the part of our models.

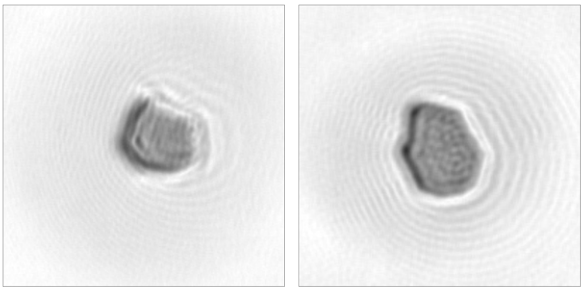


Figure 7.1: Example of two holographic orthogonal input images.

Moreover, the establishment of a reliable ground truth would facilitate the utilization of Hunyuan3D for holographic reconstruction. In the absence of such ground truth, a consensus regarding the quality of Hunyuan3D's reconstructions on the holographic images could not be reached. In contrast to our other models, which were fine-tuned by us and thus understood in greater depth, our familiarity with Hunyuan3D remained limited. The establishment of a ground truth would assist in mitigating this issue by providing a quantitative baseline for evaluation.

7.4 Leveraging Pollen Taxa

During the final phase of this work, the 3D Pollen Library was extended with taxonomic labels for each mesh. Under our two-view constraint and sparse, heterogeneous data regime, such labels are a natural prior: they could steer models toward taxon-typical morphology (e.g., spike patterns, global elongation), reducing ambiguity when imagery alone is insufficient.

With the library now tagged by overarching plant taxa, future work can explicitly condition a reconstructor on taxonomic group. In practice, this could mean feeding a learned taxon embedding

into PixelNeRF (concatenated to the encoder or MLP inputs) or modulating Pix2Vox features so the network learns taxon-typical geometry-aperture patterns, elongation, exine thickness, rather than guessing from two views alone. For mesh methods like Pixel2Mesh++, we can swap the single ellipsoid for a bank of taxon-specific mean-shape templates, selecting the template by label to start deformations closer to the right morphology.

8 Conclusion

In this thesis, we investigate whether the full geometry of individual pollen grains can be recovered from as few as two orthogonal images. Where conventional methods struggle with such sparse observations, we show that carefully selected and adapted modern models, including voxel-based approaches, mesh-deformation networks, neural radiance fields, and large diffusion transformers, can produce plausible reconstructions. Although two views rarely capture fine features or complex surface relief, our experiments indicate that, with strong implicit priors and appropriate regularization, reconstructions can be surprisingly robust and geometrically consistent. However, accuracy remains constrained by data sparsity, and claims about fine morphological detail should be made with caution.

Our comparative study, grounded in the unique 3D Pollen Library dataset, reveals that no single approach is universally superior; rather, each model family brings its own strengths and limitations. Voxel-based methods such as Pix2Vox, while robust and order-invariant, struggle to capture fine surface detail and lack the implicit priors necessary for more robust generalization. Mesh-based approaches, exemplified by Pixel2Mesh++, offer explicit surface connectivity and can recover some texture, yet are sensitive to the choice of template and prone to structural artifacts when the prior is ill-suited. Implicit neural representations, as in PixelNeRF, excel in volumetric fidelity and cross-view consistency, but their outputs may clearer recovery of textures. Large diffusion models, such as Hunyuan3D-2, has further expanded the frontier, showing that even models not explicitly trained on pollen can, with minimal adaptation, rival domain-specific architectures in terms of texture recovery, though at the cost of significant computational resources.

Throughout our experiments, we observe that the number and quality of input views plays a decisive role: while additional perspectives generally improve reconstruction, the benefit is model-dependent and not always monotonic. Notably, the transition from one to two views yields the greatest leap in fidelity, with diminishing returns thereafter. Our ablation studies on explicit priors shows that, contrary to intuition, supplying a mean-pollen template does not systematically enhance surface accuracy, though it may improve volumetric alignment if watertightness is achieved.

The journey was not without its challenges. The heterogeneity of the dataset, the need for our custom preprocessing pipelines, and the idiosyncrasies of each model's training regime necessitated a modular approach. This, in turn, led to a deeper understanding of the interplay between data, architecture, and evaluation. This was a learning process as valuable as the quantitative results themselves. Our analysis of pollen diversity further dispels the notion of universal ellipsoidal morphology and sheds light on the rich variability and complexity inherent in natural forms.

Perhaps most importantly, this work illuminates both the promise and the current limitations of automated 3D pollen reconstruction. Although, we show that meaningful shape recovery is possible under severe data constraints, the fidelity of reconstructions, especially for holographic images with minimal detail, remains bounded by the quality of input and the expressiveness of the chosen model. The bottlenecks and dead ends encountered here, show a roadmap for future research: from the integration of super-resolution and edge-preserving preprocessing, to the fine-tuning of large diffusion models on domain-specific data.

In sum, we show the feasibility of reconstructing complex biological shapes from sparse visual cues, and as a foundation for future endeavors in both computational palynology and the broader field of 3D computer vision for natural microscopic particles. We hope that these findings inform

and inspire subsequent work to ultimately enable more accurate and accessible pollen monitoring for science and society alike.

References

[1] A. Damialis, C. Traidl-Hoffmann, and R. Treudler, "Climate Change and Pollen Allergies", en, in *Biodiversity and Health in the Face of Climate Change*, M. R. Marselle, J. Stadler, H. Korn, K. N. Irvine, and A. Bonn, Eds., Cham: Springer International Publishing, 2019, pp. 47–66, ISBN: 978-3-030-02318-8. DOI: 10.1007/978-3-030-02318-8_3. [Online]. Available: https://doi.org/10.1007/978-3-030-02318-8_3 (visited on Jul. 22, 2025).

[2] E. Sauvageat, Y. Zeder, K. Auderset, *et al.*, "Real-time pollen monitoring using digital holography", English, *Atmospheric Measurement Techniques*, vol. 13, no. 3, pp. 1539–1550, Mar. 2020, Publisher: Copernicus GmbH, ISSN: 1867-1381. DOI: 10.5194/amt-13-1539-2020. [Online]. Available: https://amt.copernicus.org/articles/13/1539/2020/ (visited on Jun. 23, 2025).

[3] Y. Zhang and A. L. Steiner, "Projected climate-driven changes in pollen emission season length and magnitude over the continental United States", en, *Nature Communications*, vol. 13, no. 1, p. 1234, Mar. 2022, Publisher: Nature Publishing Group, ISSN: 2041-1723. DOI: 10.1038/s41467-022-28764-0. [Online]. Available: https://www.nature.com/articles/s41467-022-28764-0 (visited on Aug. 11, 2025).

[4] M. Sivaguru, L. Mander, G. Fried, and S. W. Punyasena, "Capturing the Surface Texture and Shape of Pollen: A Comparison of Microscopy Techniques", *PLoS ONE*, vol. 7, no. 6, e39129, Jun. 2012, ISSN: 1932-6203. DOI: 10.1371/journal.pone.0039129. [Online]. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3373610/ (visited on Jun. 23, 2025).

[5] A. Ermolaev, M. Mardini, S. Buravkov, N. Kudryavtseva, and L. Khrustaleva, "A Simple and User-Friendly Method for High-Quality Preparation of Pollen Grains for Scanning Electron Microscopy (SEM)", *Plants*, vol. 13, no. 15, p. 2140, Aug. 2024, ISSN: 2223-7747. DOI: 10.3390/plants13152140. [Online]. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC11314231/ (visited on Jun. 23, 2025).

[6] W. F. Chissoe, E. L. Vezey, and J. J. Skvarla, "Hexamethyldisilazane as a drying agent for pollen scanning electron microscopy", eng, *Biotechnic & Histochemistry: Official Publication of the Biological Stain Commission*, vol. 69, no. 4, pp. 192–198, Jul. 1994, ISSN: 1052-0295. DOI: 10.3109/10520299409106286.

[7] I. Gierlicka, I. Kasprzyk, and M. Wnuk, "Imaging Flow Cytometry as a Quick and Effective Identification Technique of Pollen Grains from Betulaceae, Oleaceae, Urticaceae and Asteraceae", *Cells*, vol. 11, no. 4, p. 598, Feb. 2022, ISSN: 2073-4409. DOI: 10.3390/cells11040598. [Online]. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC8870286/ (visited on Jun. 23, 2025).

[8] B. Mills, M. N. Zervas, and J. A. Grant-Jacob, "Pollen image manipulation and projection using latent space", English, *Frontiers in Plant Science*, vol. 16, Feb. 2025, Publisher: Frontiers, ISSN: 1664-462X. DOI: 10.3389/fpls.2025.1539128. [Online]. Available: https://www.frontiersin.org/journals/plant-science/articles/10.3389/fpls.2025.1539128/full (visited on Jun. 23, 2025).

[9] C. Wen, Y. Zhang, Z. Li, and Y. Fu, *Pixel2Mesh++: Multi-View 3D Mesh Generation via Deformation*, arXiv:1908.01491 [cs], Aug. 2019. DOI: 10.48550/arXiv.1908.01491. [Online]. Available: http://arxiv.org/abs/1908.01491 (visited on Jun. 10, 2025).

[10] A. X. Chang, T. Funkhouser, L. Guibas, *et al.*, *ShapeNet: An Information-Rich 3D Model Repository*, arXiv:1512.03012 [cs], Dec. 2015. DOI: 10.48550/arXiv.1512.03012. [Online]. Available: http://arxiv.org/abs/1512.03012 (visited on Jun. 10, 2025).

[11] C. B. Choy, D. Xu, J. Gwak, K. Chen, and S. Savarese, *3D-R2N2: A Unified Approach for Single and Multi-view 3D Object Reconstruction*, arXiv:1604.00449 [cs], Apr. 2016. DOI: 10.48550/arXiv.1604.00449. [Online]. Available: http://arxiv.org/abs/1604.00449 (visited on Jul. 15, 2025).

[12] A. Kar, C. Häne, and J. Malik, *Learning a Multi-View Stereo Machine*, arXiv:1708.05375 [cs], Aug. 2017. DOI: 10.48550/arXiv.1708.05375. [Online]. Available: http://arxiv.org/abs/1708.05375 (visited on Jul. 15, 2025).

[13] B. Mildenhall, P. P. Srinivasan, M. Tancik, J. T. Barron, R. Ramamoorthi, and R. Ng, *NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis*, arXiv:2003.08934 [cs], Aug. 2020. DOI: 10.48550/arXiv.2003.08934. [Online]. Available: http://arxiv.org/abs/2003.08934 (visited on Jul. 1, 2025).

[14] Z. Zhao, Z. Lai, Q. Lin, *et al.*, *Hunyuan3D 2.0: Scaling Diffusion Models for High Resolution Textured 3D Assets Generation*, arXiv:2501.12202 [cs], Feb. 2025. DOI: 10.48550/arXiv.2501.12202. [Online]. Available: http://arxiv.org/abs/2501.12202 (visited on Jul. 15, 2025).

[15] D. Tochilkin, D. Pankratz, Z. Liu, *et al.*, *TripoSR: Fast 3D Object Reconstruction from a Single Image*, arXiv:2403.02151 [cs], Mar. 2024. DOI: 10.48550/arXiv.2403.02151. [Online]. Available: http://arxiv.org/abs/2403.02151 (visited on Jul. 15, 2025).

[16] M. Boss, Z. Huang, A. Vasishta, and V. Jampani, *SF3D: Stable Fast 3D Mesh Reconstruction with UV-unwrapping and Illumination Disentanglement*, arXiv:2408.00653 [cs], Aug. 2024. DOI: 10.48550/arXiv.2408.00653. [Online]. Available: http://arxiv.org/abs/2408.00653 (visited on Jul. 15, 2025).

[17] J. Xu, W. Cheng, Y. Gao, X. Wang, S. Gao, and Y. Shan, *InstantMesh: Efficient 3D Mesh Generation from a Single Image with Sparse-view Large Reconstruction Models*, arXiv:2404.07191 [cs], Apr. 2024. DOI: 10.48550/arXiv.2404.07191. [Online]. Available: http://arxiv.org/abs/2404.07191 (visited on Jun. 6, 2025).

[18] R. Liu, R. Wu, B. V. Hoorick, P. Tokmakov, S. Zakharov, and C. Vondrick, *Zero-1-to-3: Zero-shot One Image to 3D Object*, arXiv:2303.11328 [cs], Mar. 2023. DOI: 10.48550/arXiv.2303.11328. [Online]. Available: http://arxiv.org/abs/2303.11328 (visited on Jun. 6, 2025).

[19] Z. Zhou and S. Tulsiani, *SparseFusion: Distilling View-conditioned Diffusion for 3D Reconstruction*, arXiv:2212.00792 [cs], Feb. 2023. DOI: 10.48550/arXiv.2212.00792. [Online]. Available: http://arxiv.org/abs/2212.00792 (visited on Jun. 6, 2025).

[20] A. Laurentini, "The visual hull concept for silhouette-based image understanding", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 16, no. 2, pp. 150–162, Feb. 1994, ISSN: 1939-3539. DOI: 10.1109/34.273735. [Online]. Available: https://ieeexplore.ieee.org/document/273735 (visited on Jun. 26, 2025).

[21] H. Xie, H. Yao, X. Sun, S. Zhou, and S. Zhang, "Pix2Vox: Context-aware 3D Reconstruction from Single and Multi-view Images", in *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, arXiv:1901.11153 [cs], Oct. 2019, pp. 2690–2698. DOI: 10.1109/ICCV.2019.00278. [Online]. Available: http://arxiv.org/abs/1901.11153 (visited on Jun. 10, 2025).

[22] N. Wang, Y. Zhang, Z. Li, Y. Fu, W. Liu, and Y.-G. Jiang, "Pixel2Mesh: Generating 3D Mesh Models from Single RGB Images", en, in *Computer Vision – ECCV 2018*, V. Ferrari, M. Hebert, C. Sminchisescu, and Y. Weiss, Eds., vol. 11215, Series Title: Lecture Notes in Computer Science, Cham: Springer International Publishing, 2018, pp. 55–71, ISBN: 978-3-030-01251-9 978-3-030-01252-6. DOI: 10.1007/978-3-030-01252-6_4. [Online]. Available: https://link.springer.com/10.1007/978-3-030-01252-6_4 (visited on Mar. 24, 2025).

[23] A. Yu, V. Ye, M. Tancik, and A. Kanazawa, *pixelNeRF: Neural Radiance Fields from One or Few Images*, arXiv:2012.02190 [cs], May 2021. DOI: 10.48550/arXiv.2012.02190. [Online]. Available: http://arxiv.org/abs/2012.02190 (visited on Jun. 10, 2025).

[24] W. Matusik, C. Buehler, R. Raskar, S. J. Gortler, and L. McMillan, "Image-based visual hulls", in *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, ser. SIGGRAPH '00, USA: ACM Press/Addison-Wesley Publishing Co., Jul. 2000, pp. 369–374, ISBN: 978-1-58113-208-3. DOI: 10.1145/344779.344951. [Online]. Available: https://dl.acm.org/doi/10.1145/344779.344951 (visited on Jun. 26, 2025).

[25] M. Li, M. Magnor, and H.-P. Seidel, "Hardware-Accelerated Visual Hull Reconstruction and Rendering", en, *Graphics Interface*, vol. 23, no. 3, Nov. 2003.

[26] K. Kutulakos and S. Seitz, "A theory of shape by space carving", in *Proceedings of the Seventh IEEE International Conference on Computer Vision*, vol. 1, Sep. 1999, 307–314 vol.1. DOI: 10.1109/ICCV.1999.791235. [Online]. Available: https://ieeexplore.ieee.org/document/791235 (visited on Jun. 26, 2025).

[27] K. Simonyan and A. Zisserman, *Very Deep Convolutional Networks for Large-Scale Image Recognition*, arXiv:1409.1556 [cs], Apr. 2015. DOI: 10.48550/arXiv.1409.1556. [Online]. Available: http://arxiv.org/abs/1409.1556 (visited on Aug. 13, 2025).

[28] O. Ronneberger, P. Fischer, and T. Brox, *U-Net: Convolutional Networks for Biomedical Image Segmentation*, arXiv:1505.04597 [cs], May 2015. DOI: 10.48550/arXiv.1505.04597. [Online]. Available: http://arxiv.org/abs/1505.04597 (visited on Aug. 13, 2025).

[29] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A large-scale hierarchical image database", in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, ISSN: 1063-6919, Jun. 2009, pp. 248–255. DOI: 10.1109/CVPR.2009.5206848. [Online]. Available: https://ieeexplore.ieee.org/document/5206848 (visited on Jul. 7, 2025).

[30] D. P. Kingma and J. Ba, *Adam: A Method for Stochastic Optimization*, arXiv:1412.6980 [cs], Jan. 2017. DOI: 10.48550/arXiv.1412.6980. [Online]. Available: http://arxiv.org/abs/1412.6980 (visited on Aug. 11, 2025).

[31] L. Ladický, O. Saurer, S. Jeong, F. Maninchedda, and M. Pollefeys, "From Point Clouds to Mesh Using Regression", in *2017 IEEE International Conference on Computer Vision (ICCV)*, ISSN: 2380-7504, Oct. 2017, pp. 3913–3922. DOI: 10.1109/ICCV.2017.420. [Online]. Available: https://ieeexplore.ieee.org/document/8237682 (visited on Jul. 8, 2025).

[32] K. He, X. Zhang, S. Ren, and J. Sun, *Deep Residual Learning for Image Recognition*, arXiv:1512.03385 [cs], Dec. 2015. DOI: 10.48550/arXiv.1512.03385. [Online]. Available: http://arxiv.org/abs/1512.03385 (visited on Aug. 13, 2025).

[33] M. Liu, C. Xu, H. Jin, *et al.*, *One-2-3-45: Any Single Image to 3D Mesh in 45 Seconds without Per-Shape Optimization*, arXiv:2306.16928 [cs], Jun. 2023. DOI: 10.48550/arXiv.2306.16928. [Online]. Available: http://arxiv.org/abs/2306.16928 (visited on Jul. 6, 2025).

[34] M. Oquab, T. Darcet, T. Moutakanni, *et al.*, *DINOv2: Learning Robust Visual Features without Supervision*, arXiv:2304.07193 [cs], Feb. 2024. DOI: 10.48550/arXiv.2304.07193. [Online]. Available: http://arxiv.org/abs/2304.07193 (visited on Aug. 13, 2025).

[35] I. Perry, J.-Y. Szeto, M. Isaacs, *et al.*, "Production of 3D printed scale models from microscope volume datasets for use in STEM education", en, *EMS Engineering Science Journal*, vol. 1, no. 1, Jul. 2017, Number: 1 Publisher: EMS Publishing. [Online]. Available: https://orca.cardiff.ac.uk/id/eprint/104773/1/EMS%20Eng%20Sci%20j%202%202017.pdf (visited on Jun. 2, 2025).

[36] W. E. Lorensen and H. E. Cline, "Marching cubes: A high resolution 3D surface construction algorithm", *SIGGRAPH Comput. Graph.*, vol. 21, no. 4, pp. 163–169, Aug. 1987, ISSN: 0097-8930. DOI: 10.1145/37402.37422. [Online]. Available: https://dl.acm.org/doi/10.1145/37402.37422 (visited on Aug. 13, 2025).

[37] Y. Chen and G. Medioni, "Object modeling by registration of multiple range images", in *1991 IEEE International Conference on Robotics and Automation Proceedings*, Apr. 1991, 2724–2729 vol.3. DOI: 10.1109/ROBOT.1991.132043. [Online]. Available: https://ieeexplore.ieee.org/document/132043 (visited on Jun. 24, 2025).

[38] S. Rusinkiewicz and M. Levoy, "Efficient variants of the ICP algorithm", in *Proceedings Third International Conference on 3-D Digital Imaging and Modeling*, May 2001, pp. 145–152. DOI: 10.1109/IM.2001.924423. [Online]. Available: https://ieeexplore.ieee.org/document/924423 (visited on Jun. 24, 2025).

[39] T. Huang, Q. Liu, X. Zhao, J. Chen, and Y. Liu, *Learnable Chamfer Distance for Point Cloud Reconstruction*, arXiv:2312.16582 [cs], Dec. 2023. DOI: 10.48550/arXiv.2312.16582. [Online]. Available: http://arxiv.org/abs/2312.16582 (visited on Jul. 8, 2025).

[40] T. Wu, L. Pan, J. Zhang, T. Wang, Z. Liu, and D. Lin, *Density-aware Chamfer Distance as a Comprehensive Metric for Point Cloud Completion*, arXiv:2111.12702 [cs], Nov. 2021. DOI: 10.48550/arXiv.2111.12702. [Online]. Available: http://arxiv.org/abs/2111.12702 (visited on Jul. 8, 2025).

[41] P. E. Jardine, L. Palazzesi, M. C. Tellería, and V. D. Barreda, "Why does pollen morphology vary? Evolutionary dynamics and morphospace occupation in the largest angiosperm order (Asterales)", en, *New Phytologist*, vol. 234, no. 3, pp. 1075–1087, 2022, _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/nph.18024, ISSN: 1469-8137. DOI: 10.1111/nph.18024. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1111/nph.18024 (visited on Jun. 4, 2025).

[42] H. Halbritter, S. Ulrich, F. Grímsson, *et al.*, *Illustrated Pollen Terminology*, en. Cham: Springer International Publishing, 2018, ISBN: 978-3-319-71364-9 978-3-319-71365-6. DOI: 10.1007/978-3-319-71365-6. [Online]. Available: https://link.springer.com/10.1007/978-3-319-71365-6 (visited on Jun. 2, 2025).

[43] K. Hiroi, M. Sone, S. Sakazono, *et al.*, "Time-lapse imaging of self- and cross-pollination in Brassica rapa", *Annals of botany*, vol. 112, May 2013. DOI: 10.1093/aob/mct102.

[44] M. Deitke, R. Liu, M. Wallingford, *et al.*, *Objaverse-XL: A Universe of 10M+ 3D Objects*, arXiv:2307.05663 [cs], Jul. 2023. DOI: 10.48550/arXiv.2307.05663. [Online]. Available: http://arxiv.org/abs/2307.05663 (visited on Jun. 10, 2025).

[45] A. Radford, J. W. Kim, C. Hallacy, *et al.*, *Learning Transferable Visual Models From Natural Language Supervision*, arXiv:2103.00020 [cs], Feb. 2021. DOI: 10.48550/arXiv.2103.00020. [Online]. Available: http://arxiv.org/abs/2103.00020 (visited on Aug. 5, 2025).

[46] R. Xu, L. Liu, N. Wang, *et al.*, *CWF: Consolidating Weak Features in High-quality Mesh Simplification*, arXiv:2404.15661 [cs], Apr. 2024. DOI: 10.48550/arXiv.2404.15661. [Online]. Available: http://arxiv.org/abs/2404.15661 (visited on Aug. 13, 2025).

[47] M. Kazhdan and H. Hoppe, "Screened poisson surface reconstruction", en, *ACM Transactions on Graphics*, vol. 32, no. 3, pp. 1–13, Jun. 2013, ISSN: 0730-0301, 1557-7368. DOI: 10.1145/2487228.2487237. [Online]. Available: https://dl.acm.org/doi/10.1145/2487228.2487237 (visited on Jun. 2, 2025).

[48] N. Otsu, "A Threshold Selection Method from Gray-Level Histograms", *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 9, no. 1, pp. 62–66, Jan. 1979, ISSN: 2168-2909. DOI: 10.1109/TSMC.1979.4310076. [Online]. Available: https://ieeexplore.ieee.org/document/4310076 (visited on Aug. 1, 2025).

Declaration of Authenticity

I hereby declare that any individual work / pair work / team work submitted for assessment is entirely the product of my own / my own and my partner's / my own and my team's effort,

- that I/we have correctly cited all text passages that do not originate from me/us, in accordance with standard academic citation rules[3], and that I/we have clearly mentioned all sources used;
- that I/we have declared in footnotes or in an index of auxiliary tools all aids used (AI assistance systems such as chatbots[4], translation[5], paraphrasing[6], or programming applications[7], and indicated their use at the corresponding text passages;
- that I/we have acquired all intangible rights to any materials I/we may have used, such as images or graphics, or that these materials were created by me/us;
- that the topic, the thesis or parts of it have not been used in an assessment of another module, unless this has been expressly agreed with the lecturer in advance and is stated as such;
- that I/we am/are aware that my/our work may be checked for plagiarism and for third-party authorship of human or technical origin (artificial intelligence);
- that I/we am/are aware that the FHNW School of Engineering will pursue a violation of this declaration of authenticity and that disciplinary consequences (reprimand or expulsion from the study program) may result from this.

Windisch, 14. August 2025

Name: Nils Fahrni

Signature:

Name: Etienne Roulet

Signature:

[3] e.g. APA or IEEE
[4] e.g., ChatGPT
[5] e.g., DeepL
[6] e.g., Quillbot
[7] e.g., Github Copilot

A Appendix

A.1 Vanilla NeRF Experiment

This section outlines the implementation of a vanilla NeRF experiment designed to assess the effect of the number of training views on the reconstruction quality. It compares the performance of NeRF models that have been trained using different numbers of input views (2, 3, 4, 5, 6, 7, 8, 60, 80 and 100) across multiple object instances. This experiment forms the basis of our thesis that NeRF and other models do not work in a setting with two image views.

A.1.1 Experimental Setup

The vanilla NeRF implementation includes the following key components:

- **Model Architecture**: Standard NeRF with 128-dimensional hidden layers
- **Training Parameters**: 3 epochs, learning rate 0.0001, batch size 1024
- **Rendering Parameters**: Near bound $t_n = 6.0$, far bound $t_f = 12.0$, 128 sampling bins
- **Evaluation**: Global PSNR computed over all test pixels from 100-view test set
- **Point Cloud Extraction**: Marching cubes applied to density field on 128^3 grid

A.1.2 Experimental Methodology

To investigate the impact of data sparsity on the vanilla NeRF reconstruction quality, we conducted a systematic study by varying the number of training views across multiple object instances. The experiment was designed to demonstrate how the performance of vanilla NeRF degrades with limited training data, highlighting one of the key limitations that our proposed methods aim to address.

Training Data Sparsity Investigation: We trained separate NeRF models using different numbers of input views: 2, 3, 4, 5, 6, 7, 8, 60, 80, and 100. Low view counts (2-8) represent extremely sparse training conditions, whereas higher counts (60-100) provide more comprehensive coverage. This range allowed us to observe the transition from severely under-constrained to well-constrained scenarios.

Evaluation Protocol: All models were evaluated on a consistent test set derived from 100 novel views to ensure a fair comparison across different training conditions. The global PSNR was computed across all test pixels to provide a comprehensive quality metric that captures both the structural accuracy and color fidelity.

3D Geometry Assessment: Beyond novel view synthesis metrics, we extracted 3D point clouds from the learned density fields using marching cubes on a 128^3 spatial grid. This visualisation approach reveals how geometric understanding deteriorates under sparse view conditions, where the vanilla NeRF frequently fails to reconstruct coherent 3D structures.

Multi-Object Validation: The experiment was conducted using four different pollen grain specimens (IDs: 17818, 17900, 20938, and 17822) to ensure that the observed performance patterns were consistent across different object geometries and not artefacts of a single test case.

A.1.3 Sparsity Problem Demonstration

This experiment serves as a baseline to demonstrate the fundamental limitations of vanilla NeRF under sparse-view conditions, providing motivation for the advanced methods presented in this thesis:

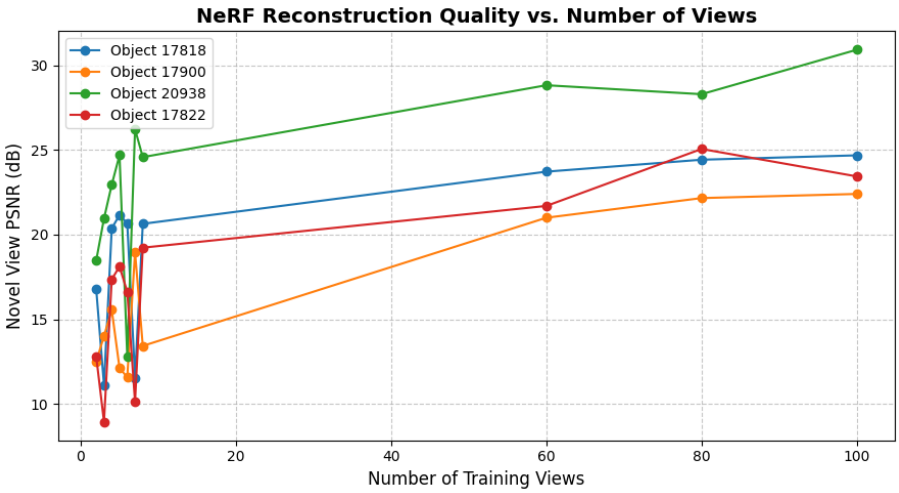


Figure A.1: Vanilla NeRF number of views Plot

- **Severe Performance Degradation**: With only 2-8 training views, vanilla NeRF exhibits dramatically reduced reconstruction quality, often failing to learn coherent 3D geometry. This demonstrates the dependence of the method on dense-view sampling for effective scene representation.
- **Geometric Inconsistencies**: Under sparse conditions, the extracted point clouds reveal fragmented and incomplete 3D structures, highlighting how insufficient view coverage leads to poor density field learning and geometric hallucinations.
- **Limited Generalization**: The poor performance on novel views when trained with sparse data illustrates vanilla NeRF's inability to interpolate effectively between widely spaced viewpoints, a critical limitation for practical applications.
- **Motivation for Alternative Approaches**: The results demonstrate why traditional NeRF is unsuitable for scenarios with limited training data, such as pollen reconstruction where obtaining dense view coverage is impractical or impossible.

The experiment conclusively shows that vanilla NeRF's requirement for dense view sampling presents a significant barrier to practical deployment, necessitating the development of more data-efficient 3D reconstruction methods, as explored in this thesis.

Result The following two plots illustrate how the reconstruction quality of NeRF depends on the number of input views. In our tests, a minimum of 60 images was required to obtain a usable mesh.

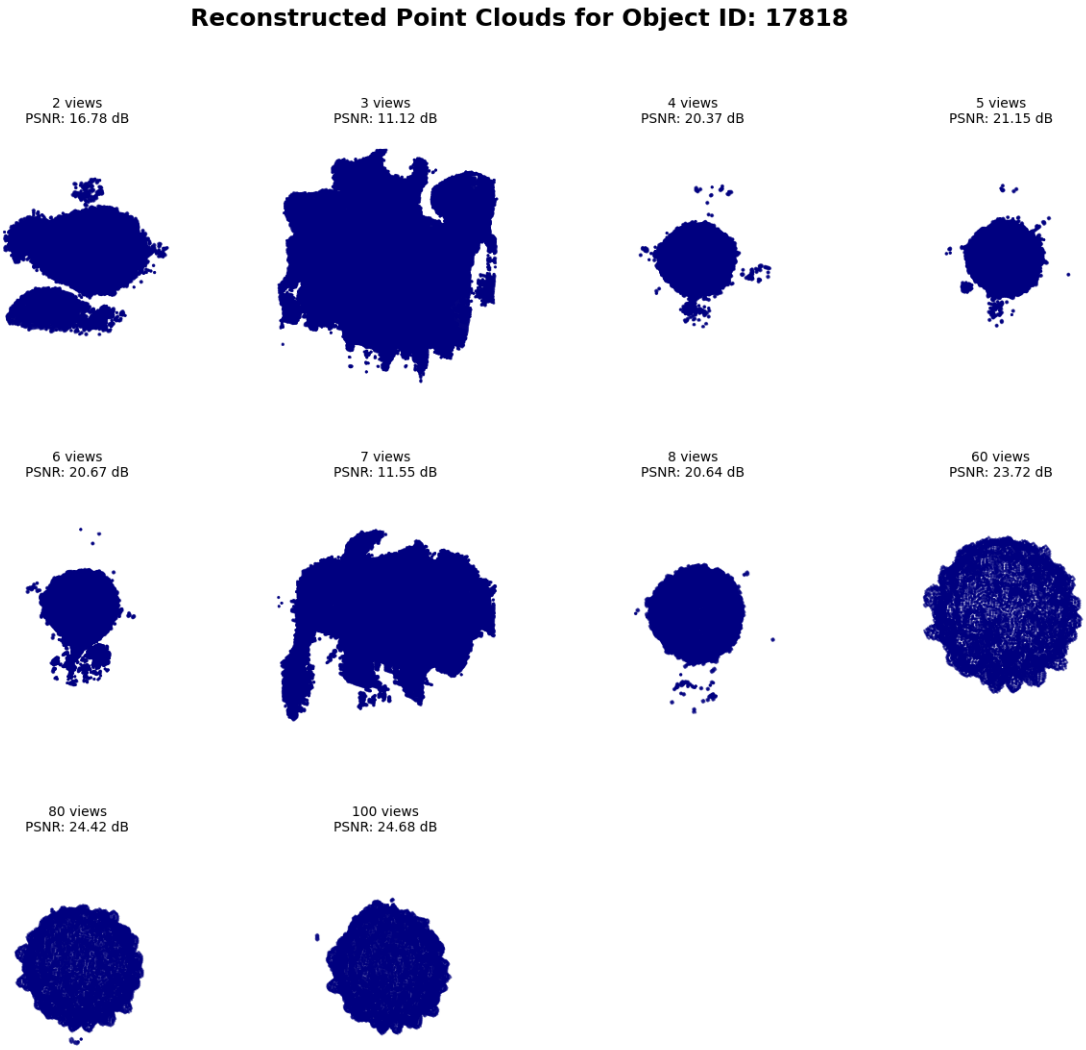


Figure A.2: Experiment Vanilla NeRF - Simple Pollen Structure

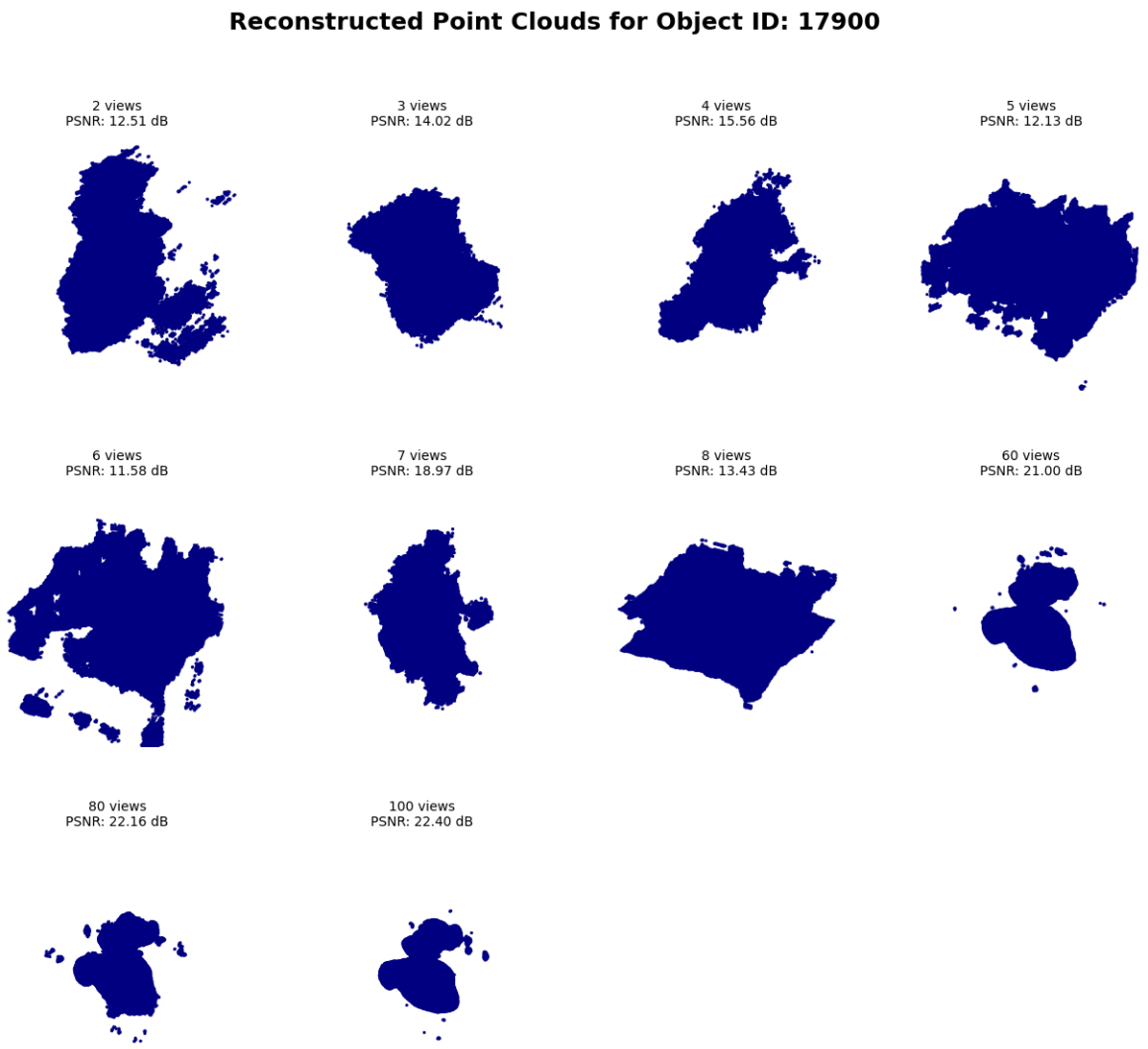


Figure A.3: Experiment Vanilla NeRF - Complex Pollen Structure

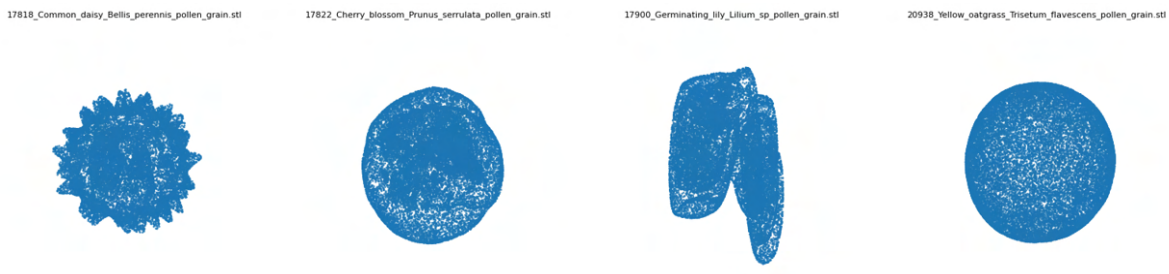


Figure A.4: Ground Truth Vanilla NeRF

Implementation details see: Vanilla NeRF Notebook

A.2 Experiment 2: Accompanying Results

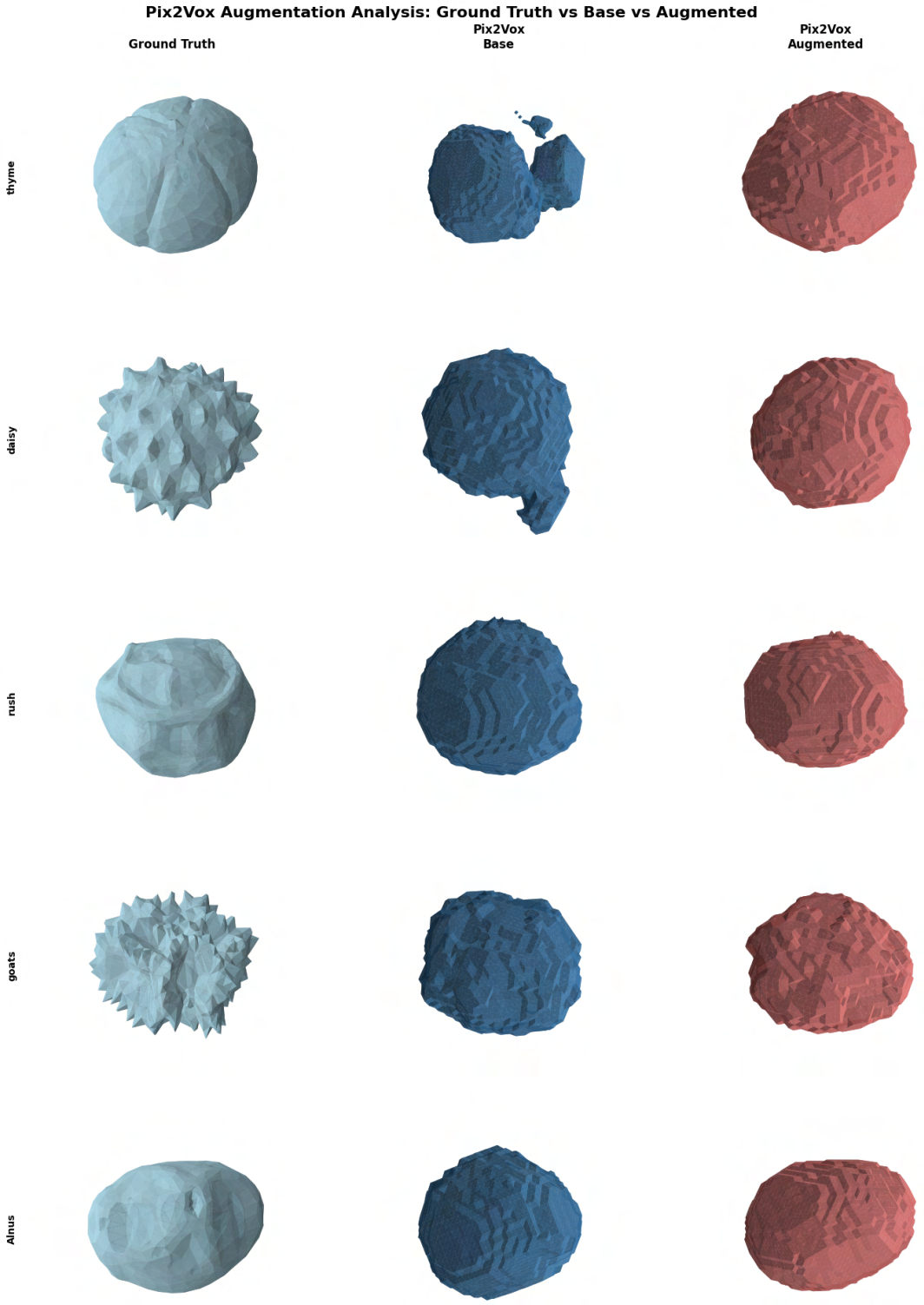


Figure A.5: The utilization of augmentations in training has been demonstrated to yield substantial success in the Pix2Vox process. The incorporation of augmentation data has been shown to enhance the clarity of the noise such as tails and other random spikes, resulting in a more pronounced overlap with the ground truth.

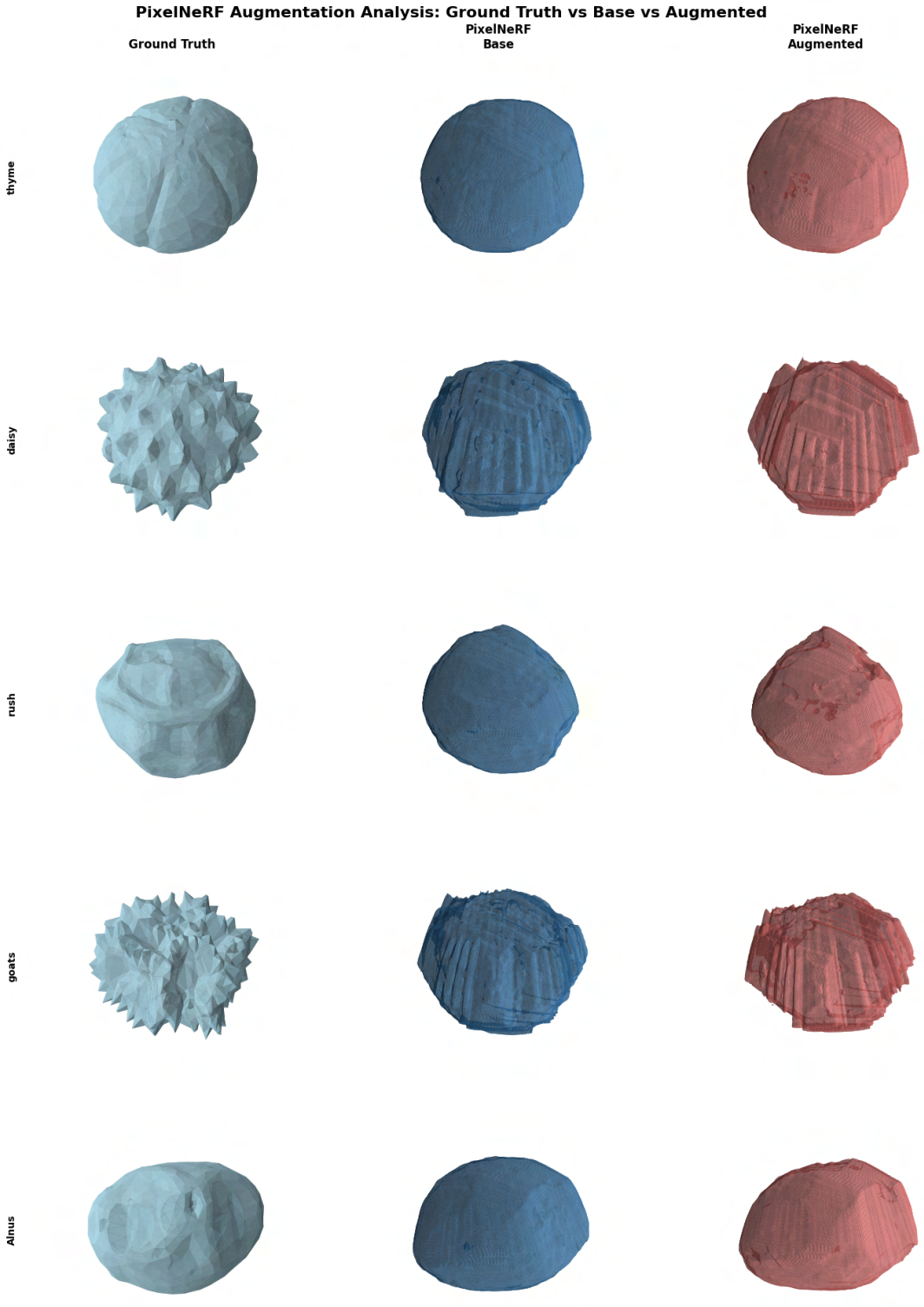


Figure A.6: The success of augmentation in PixelNeRF is more subtle and not immediately apparent. Instead, the quantitative metrics, such as chamfer loss, are more robust.

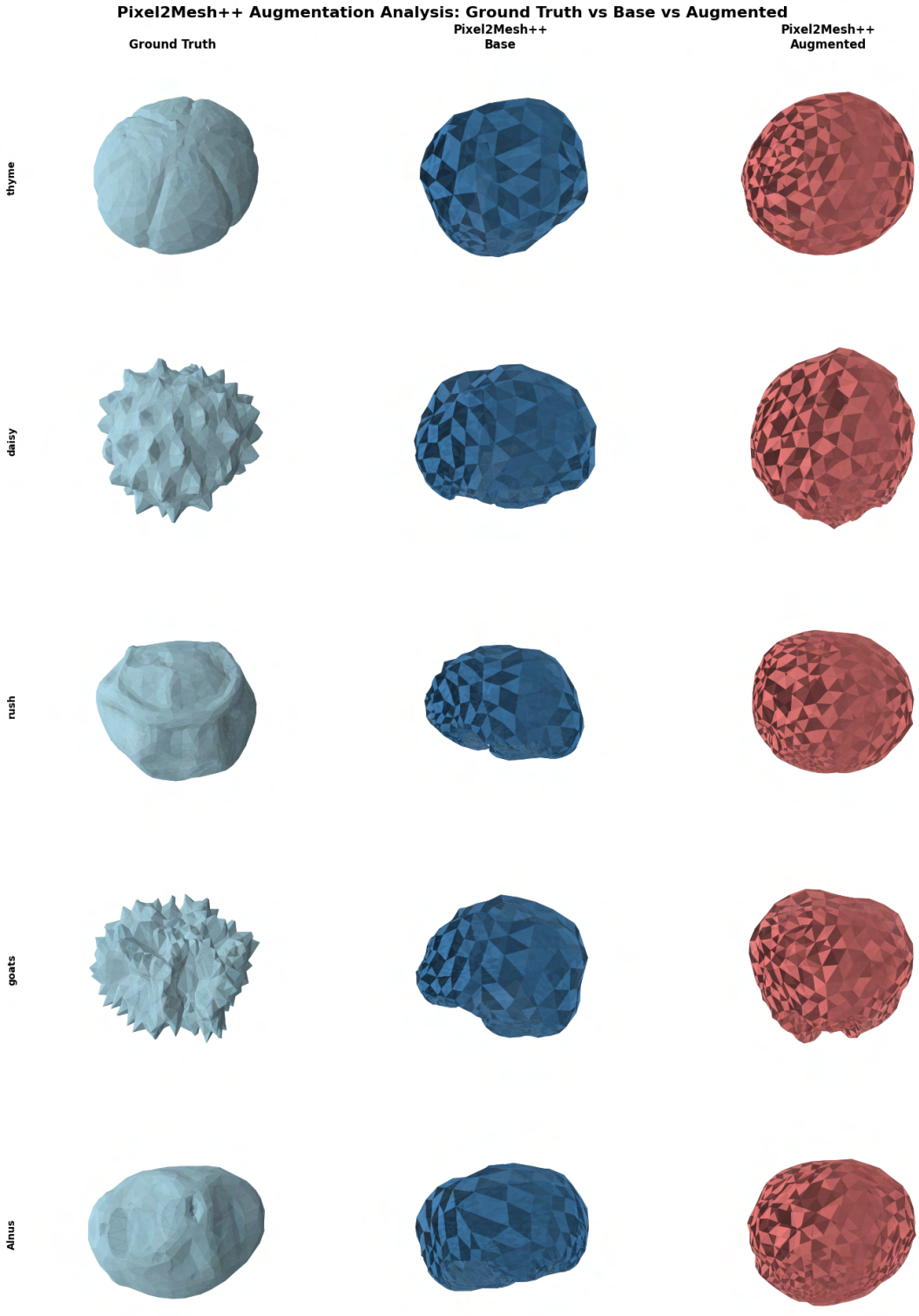


Figure A.7: Initially, the model that was trained on the original, in conjunction with the augmented pollen, appears to be inferior to the model without augmented pollen. However, this is due to the fact that only one side of the 3D model is visible. To perform a more robust comparison, it is necessary to compare all sides. This would reveal significant improvements in the chamfer distance, as well as a greater standard deviation.

A.3 Experiment 5: Accompanying Results

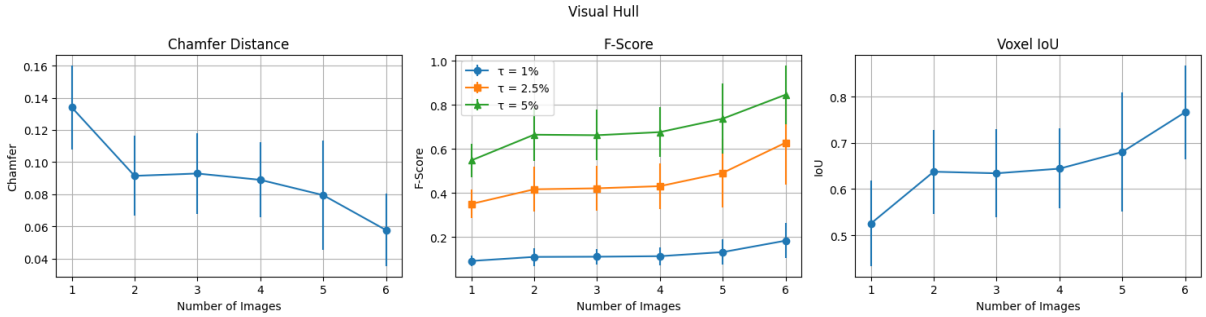


Figure A.8: Visual Hull Quantitative Results plotted.

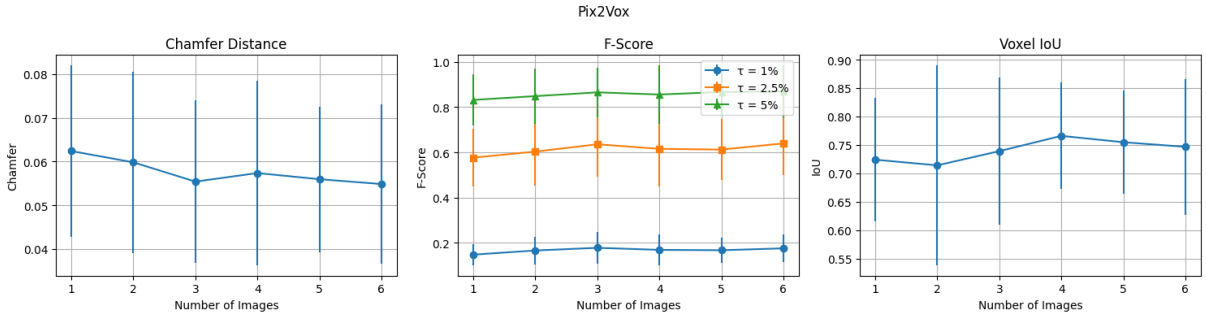


Figure A.9: Pix2Vox Quantitative Results plotted.

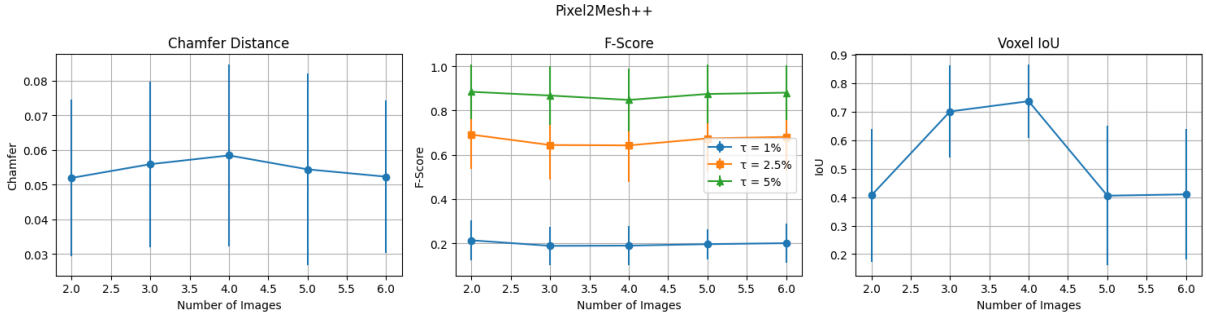


Figure A.10: Pixel2Mesh Quantitative Results plotted.

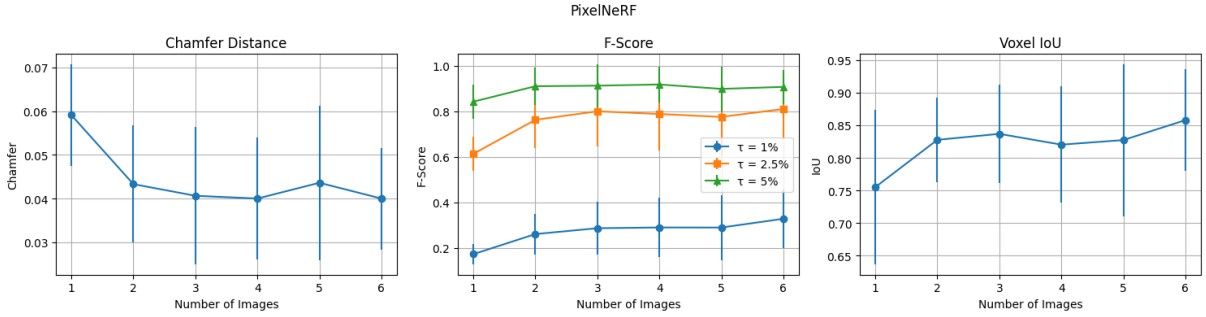


Figure A.11: PixelNeRF Quantitative Results plotted.

A.4 Hyperparameter Tuning PixelNeRF

A.4.1 Resolution

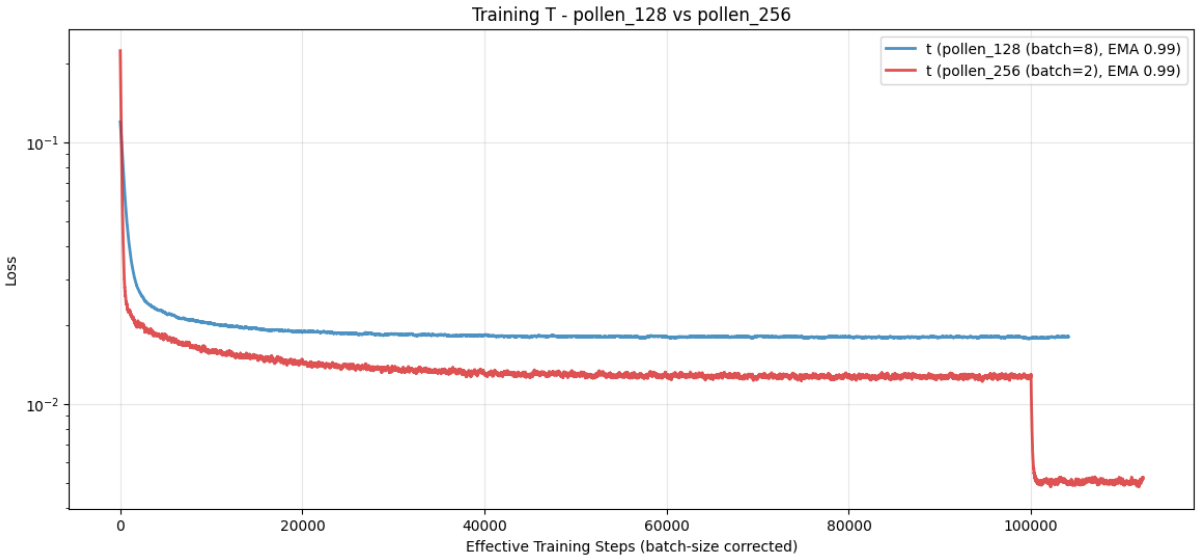


Figure A.12: Comparison of PixelNeRF total loss for 128 vs. 256 resolution. Bounding-box sampling is disabled after 100k iterations. Due to the corrected batch size, the effective training steps for pollen 128 amount to only around 20k.

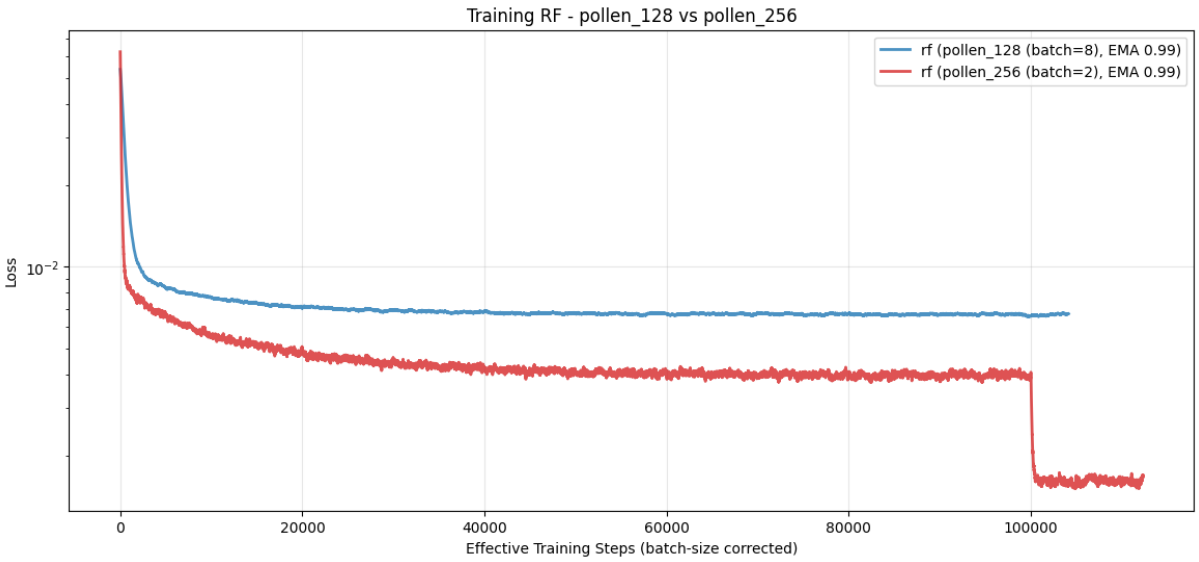


Figure A.13: PixelNeRF 128 vs 256 loss r-fine

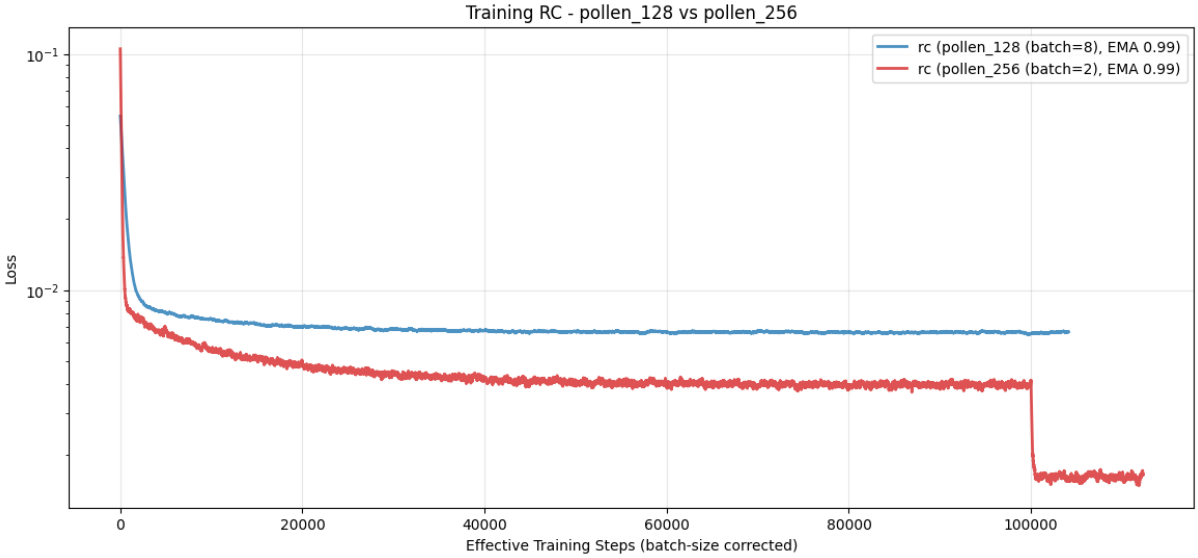


Figure A.14: PixelNeRF 128 vs 256 loss r-coarse

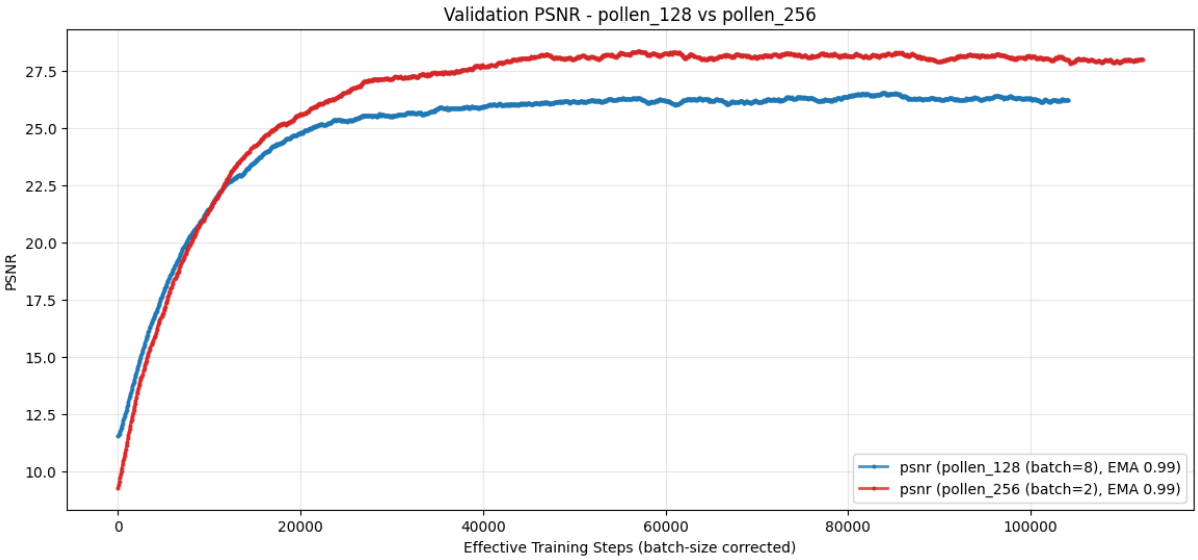


Figure A.15: PixelNeRF 128 vs 256 loss psnr

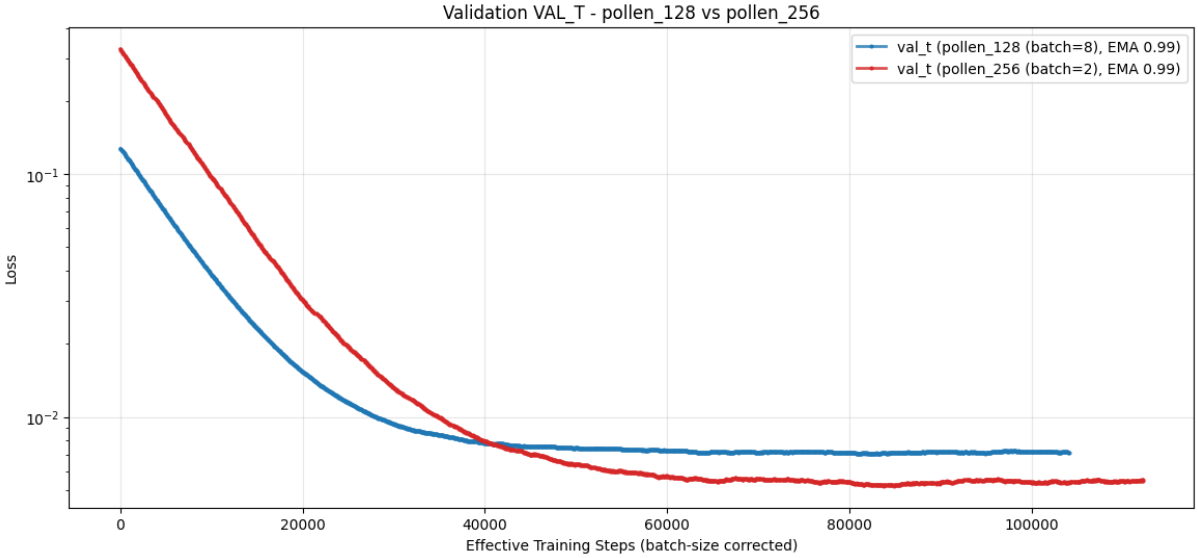


Figure A.16: PixelNeRF 128 vs 256 val loss total

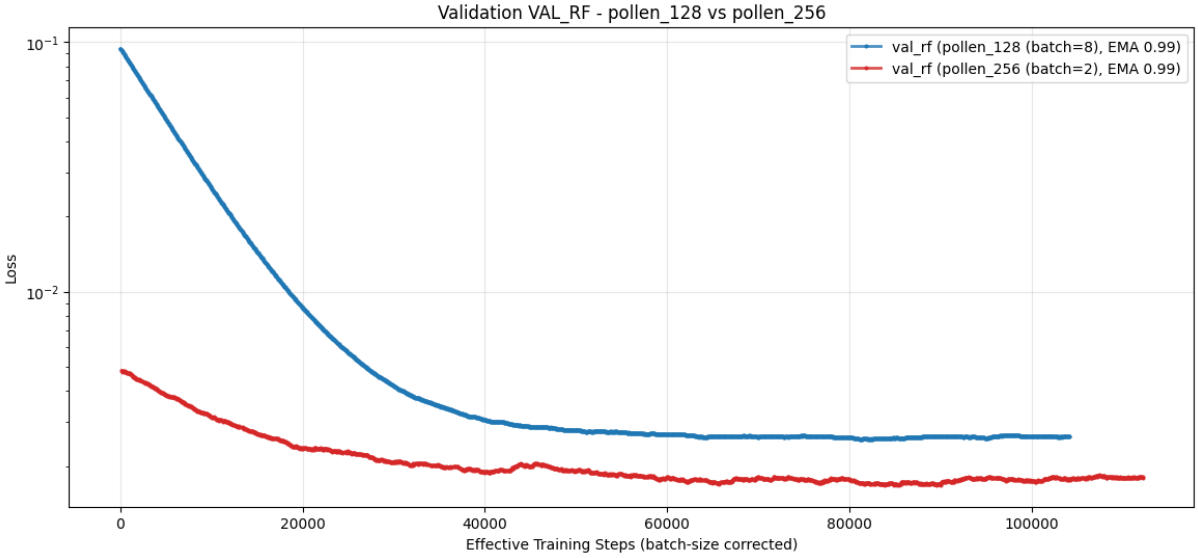


Figure A.17: PixelNeRF 128 vs 256 val loss r-fine

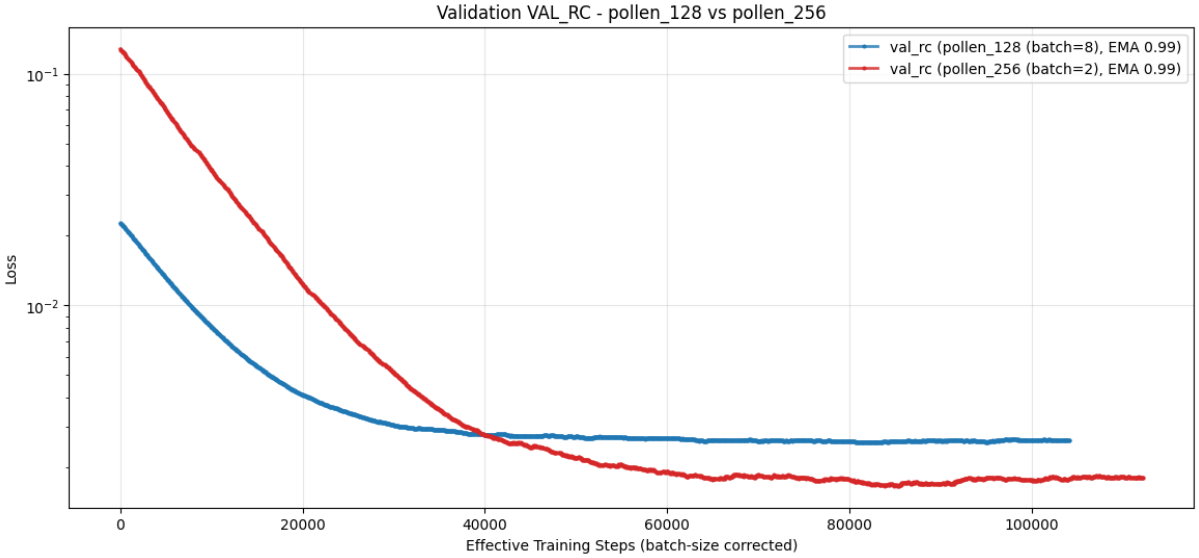


Figure A.18: PixelNeRF 128 vs 256 val loss r-coarse

A.4.2 Nerf Parameters

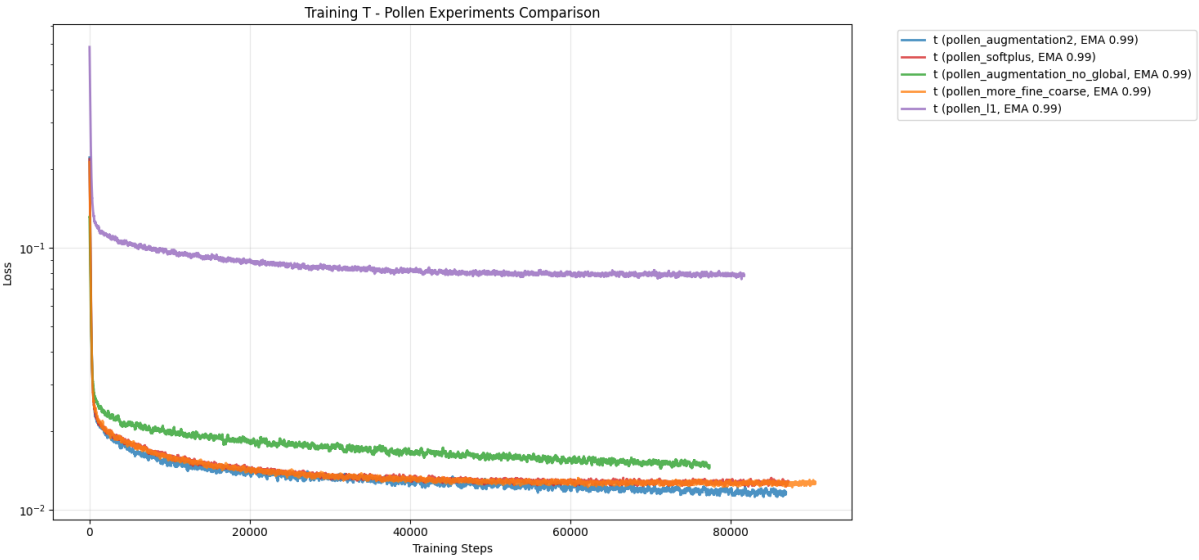


Figure A.19: PixelNeRF parameters loss total

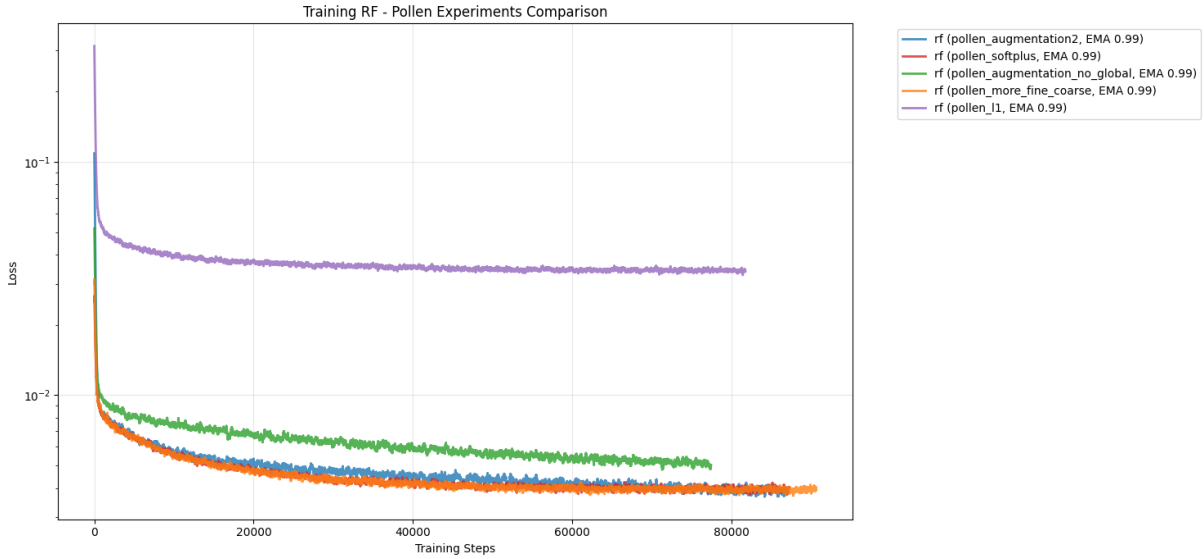


Figure A.20: PixelNeRF parameters loss r-fine

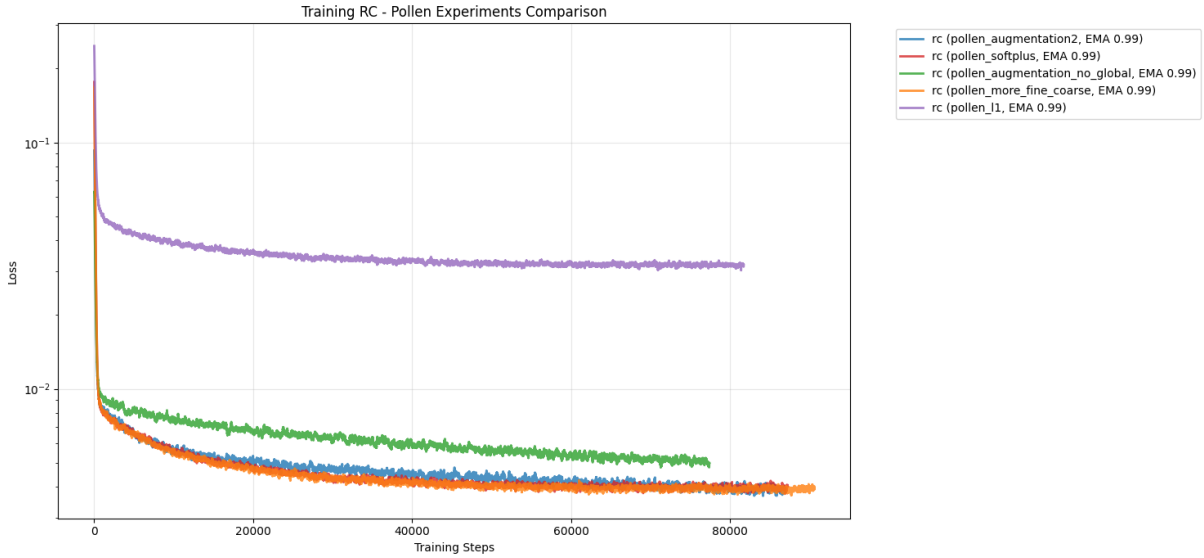


Figure A.21: PixelNeRF parameters loss r-coarse

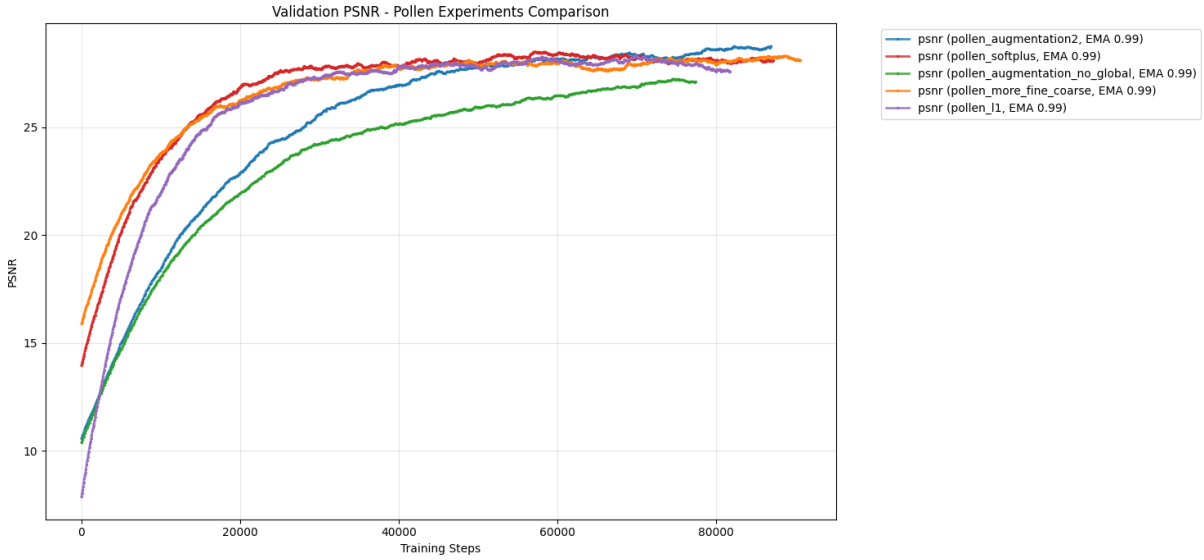


Figure A.22: PixelNeRF parameters val psnr

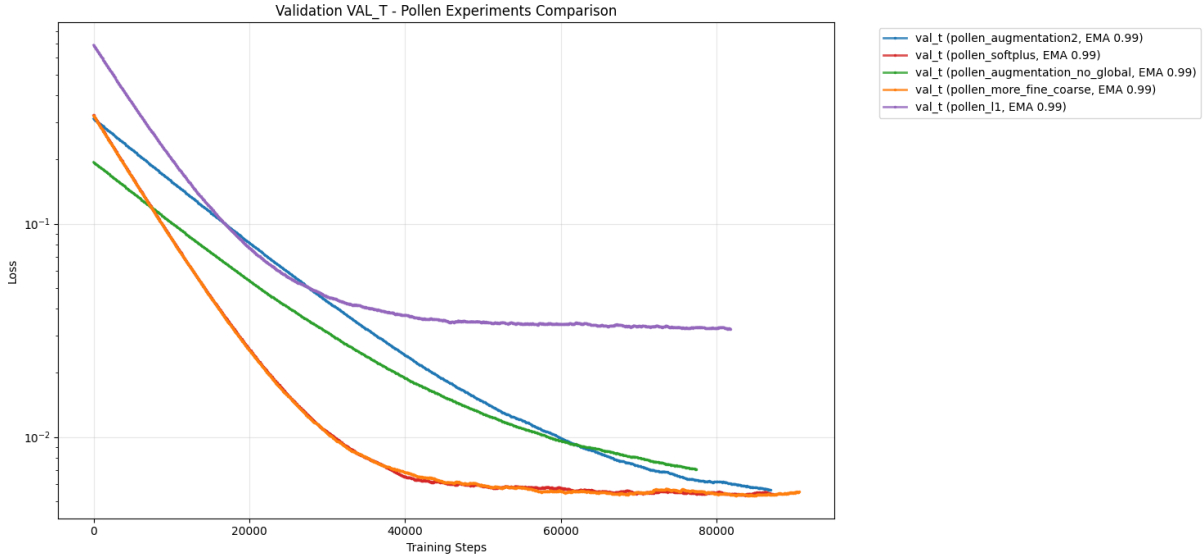


Figure A.23: PixelNeRF parameters val loss total

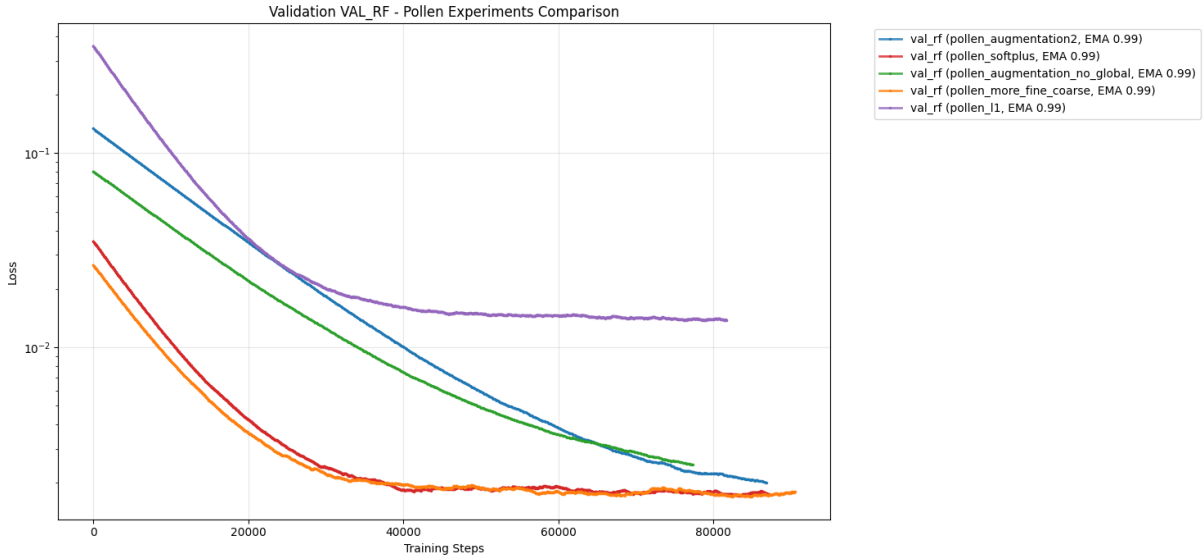


Figure A.24: PixelNeRF parameters val loss r-fine

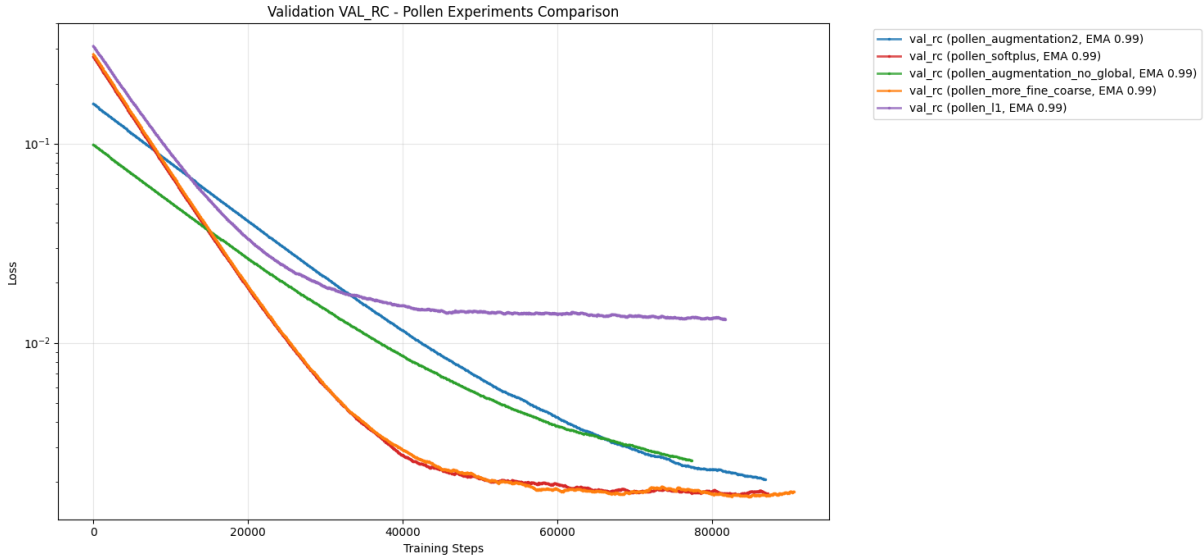


Figure A.25: PixelNeRF parameters val loss r-coarse

Reproduction of PixelNeRF Hyperparameters:

Table A.1: Key hyperparameter configurations for PixelNeRF parameters.

Parameter	Pollen (Baseline)	128px	Augmented	Augmented (L1)	Augmented (No Global)	Augmented (Big Enc)	Augmented (More PE)	Scratch
Image Resolution	256x256	128x128	256x256	256x256	256x256	256x256	256x256	256x256
Global Encoder	False	False	True	True	False	True	True	False
Encoder Backbone	ResNet-34	ResNet-34	ResNet-34	ResNet-34	ResNet-34	ResNet-50	ResNet-34	ResNet-34
Encoder Pretrained	True	True	True	True	True	True	True	False
Pos. Enc. Freqs.	6	6	6	6	6	6	10	8
Coarse Samples	64	64	128	128	128	128	128	64
Fine Samples	32	32	64	64	64	64	64	32
L1 Loss	False	False	False	True	False	False	False	True
Alpha Regularization	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001

For more Hyperparameter Tuning see PixelNeRF Project on Wandb

A.5 Hyperparameter Tuning Hunyuan3D-2

Quantitative Results 5 Inference Steps Octree380

Model	Chamfer	F-Score			IoU
		$\tau_1 = 1\%$	$\tau_2 = 2.5\%$	$\tau_3 = 5\%$	
Hunyuan3D-two-views	0.061 ± 0.033	18.2 ± 8.3	60.8 ± 16.0	84.5 ± 13.0	70.8 ± 13.4
Hunyuan3D-three-views	0.049 ± 0.014	20.9 ± 9.8	67.6 ± 13.8	88.9 ± 8.6	74.97 ± 11.8
Hunyuan3D-four-views	0.054 ± 0.026	21.0 ± 9.0	66.8 ± 15.7	86.9 ± 12.0	73.5 ± 14.1

Table A.2: Validation metrics for all Hunyuan3D-2 variants on the validation partition of the 3D Pollen Library dataset.

Quantitative Results Inference Steps 50 Octree

Model	Chamfer	F-Score			IoU
		$\tau_1 = 1\%$	$\tau_2 = 2.5\%$	$\tau_3 = 5\%$	
Hunyuan3D	0.0477 ± 0.0148	22.68 ± 10.09	69.68 ± 14.38	89.30 ± 8.66	76.02 ± 10.38
Hunyuan3D-four-views	0.0443 ± 0.0129	25.72 ± 9.81	73.55 ± 12.72	90.50 ± 7.63	77.89 ± 9.90
Hunyuan3D-two-views	0.0432 ± 0.0126	25.93 ± 8.69	75.18 ± 12.47	91.15 ± 7.80	79.05 ± 9.58

Table A.3: Validation metrics for Hunyuan3D-2 variants on the validation partition of the 3D Pollen Library dataset (50 inference steps, octree resolution 380).

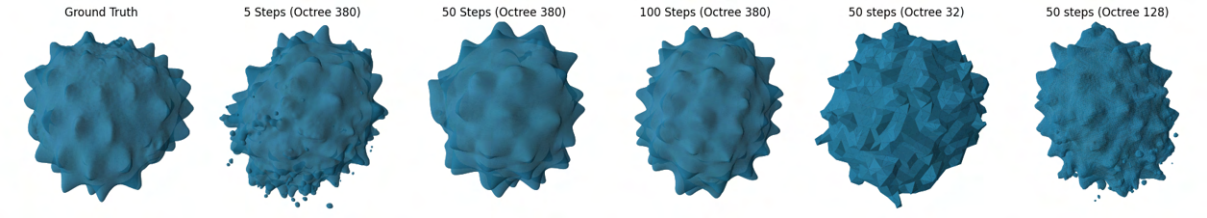


Figure A.26: Hunyuan3D-2 Overview of Hyperparameters

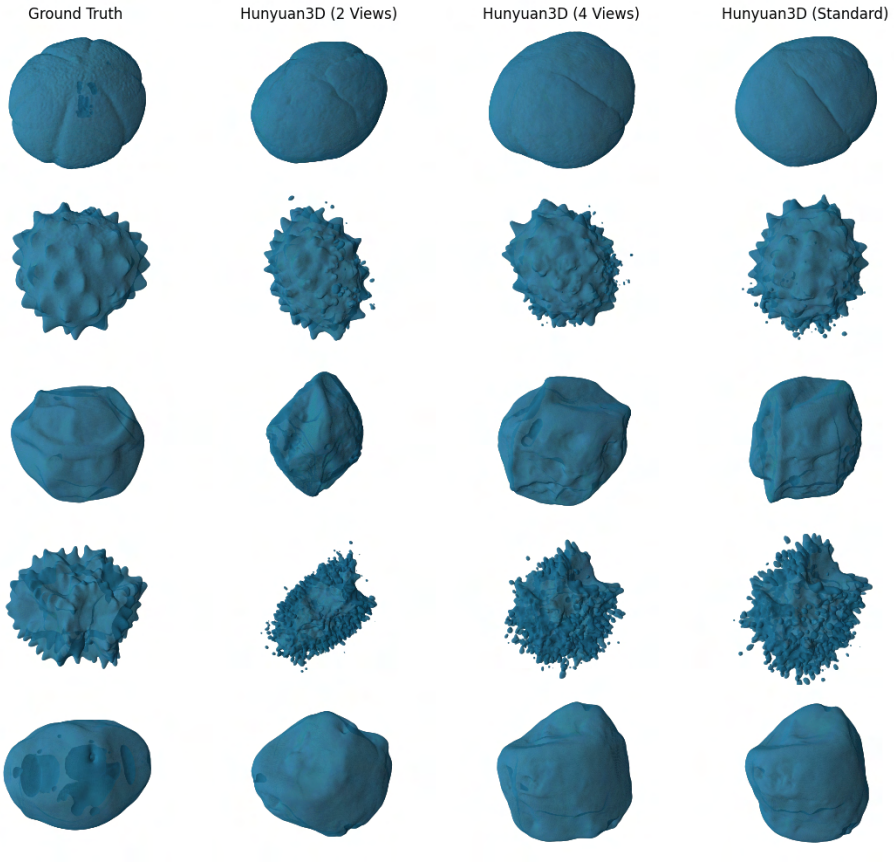


Figure A.27: Hunyuan3D-2 Reconstructions 5 Inference Steps

Figure A.28: Reconstruction quality of Hunyuan3D on complex, fine-structured pollen grains degrades at lower inference-step counts and is substantially improved when using a greater number of inference steps.

A.6 Exploratory Data Analysis

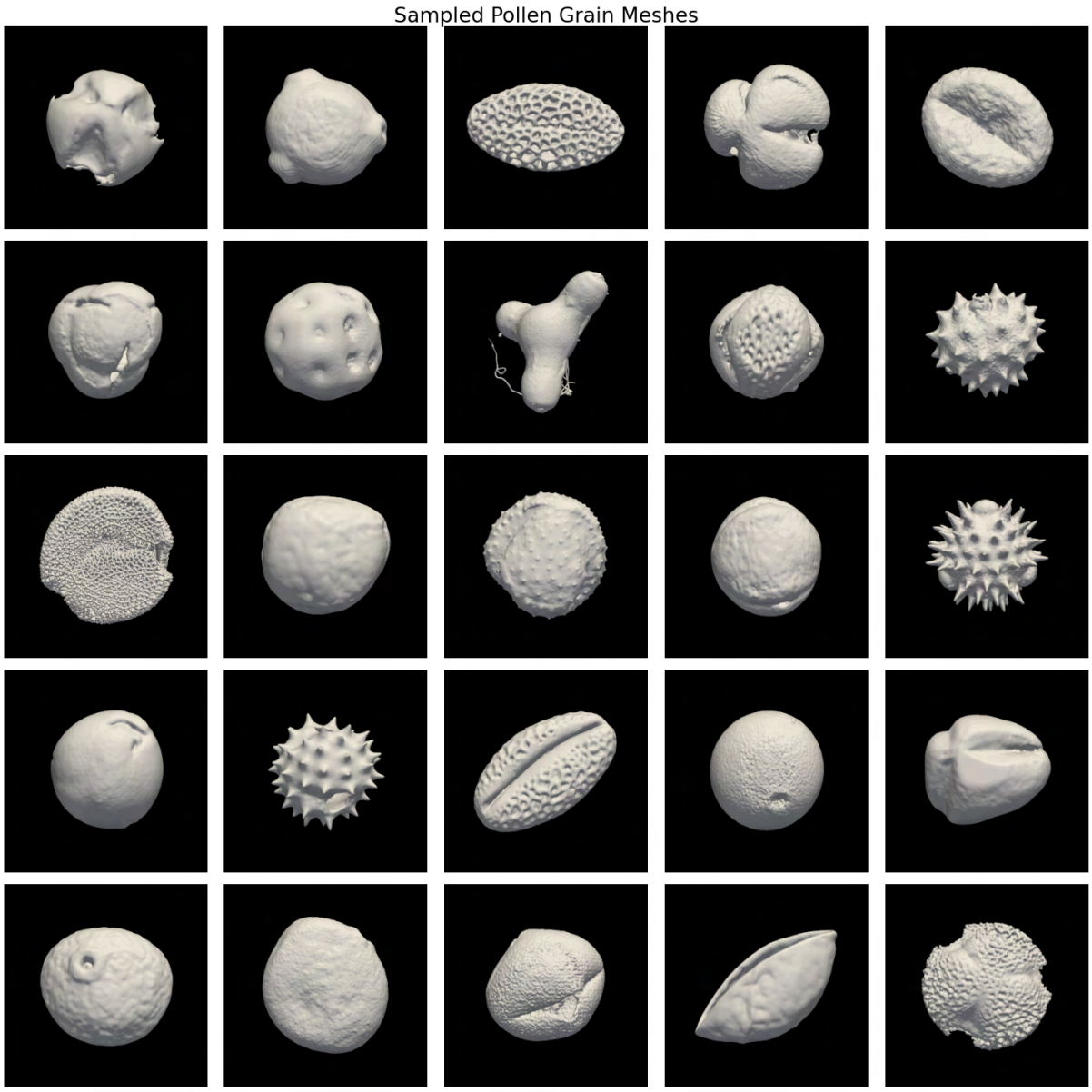


Figure A.29: An Overview of different Pollen Shapes

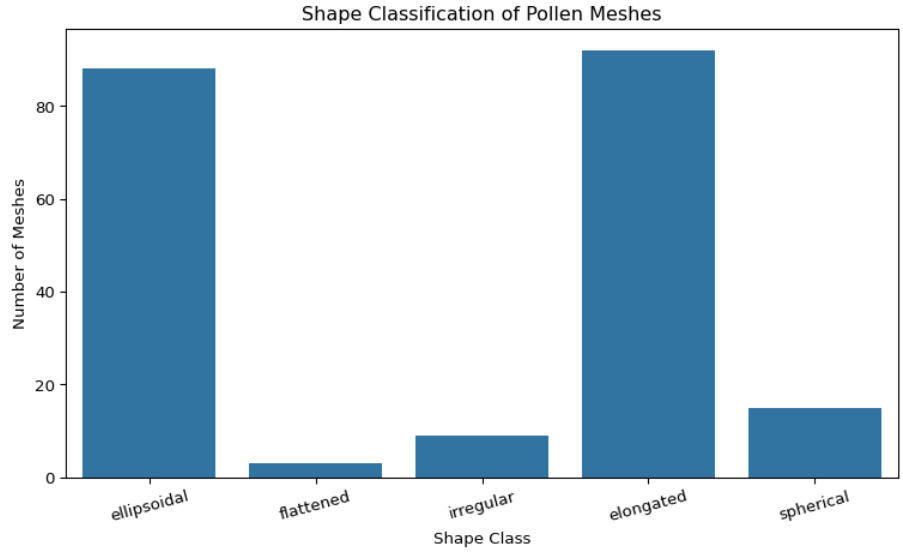


Figure A.30: Pollen Shape Classification

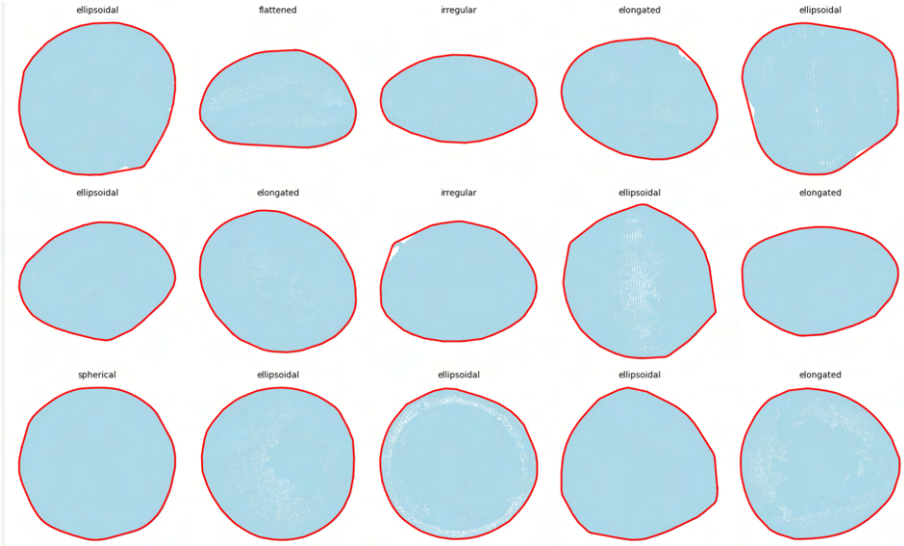


Figure A.31: Different Pollen Shapes Part 1

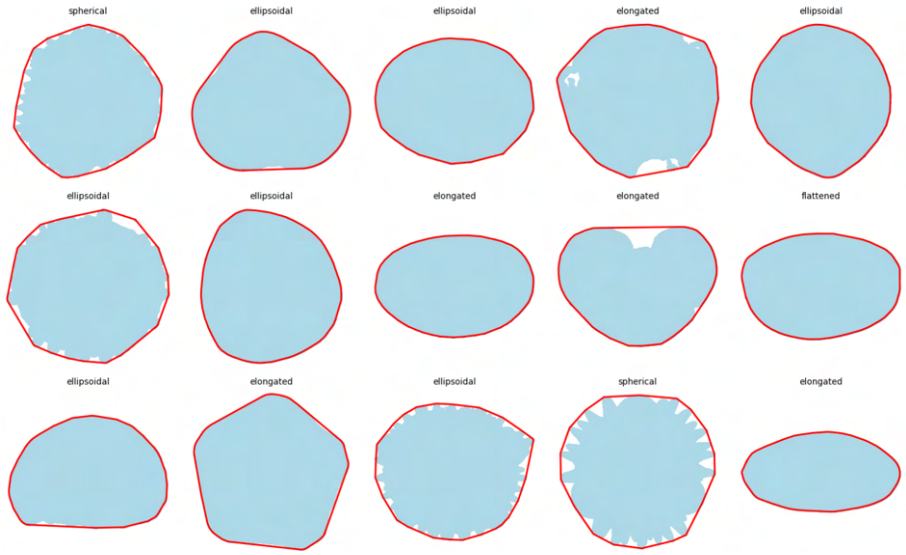


Figure A.32: Different Pollen Shapes Part 2

A.7 Constructing Priors for Pixel2Mesh++

Since Pixel2Mesh [22] (and therefore Pixel2Mesh++ [9]) both set up their GCN architecture to only handle 156-vertex template meshes, we follow the requirement of constructing priors that have 156 vertices. Moreover, due to the architectural constraint (fixed GCN input parameter count of 156), we cannot pass in any arbitrary prior to Pixel2Mesh++, it has to stay the same for every sample during training, validation and test/inference time.

While many mesh-simplification algorithms allow to specify a target number of vertices, in practice they do not guarantee achieving that exact count [46]. Most techniques, such as edge-collapse decimation guided by quadric error metrics, reduce vertex or face counts in discrete steps, and cannot always land precisely on the desired target (e.g., removal of an internal edge deletes two triangles, while collapsing a boundary edge may remove just one; this leads to parity effects and unpredictable counts). Fundamentally, this stems from the combinatorial and graph-theoretic nature of meshes: simplification proceeds via local connectivity operations, not continuous or count-preserving transformations. Consequently, although algorithms can be guided toward a desired count, they typically only approximate it, rather than deterministically hit it exactly.

A.7.1 Constructing a Mean Prior with 156 vertices

Given a set of binary voxel grids $\{V_i\}_{i=1}^{N}$ (each $V_i \in \{0,1\}^{X\times Y\times Z}$), we form a consensus occupancy by stacking the tensors and taking a per-voxel mean $\bar{V} = \frac{1}{N}\sum_i V_i$. We then apply a majority-vote threshold at 0.5 (i.e., $\hat{V} = \mathbb{1}[\bar{V} \geq 0.5]$) to obtain a "mean shape" that retains voxels present in the training exemplars. Majority voting is a standard and well-studied label-fusion strategy for aggregating multiple segmentations, prized for its simplicity and robustness in practice.[8] From this binary volume $\hat{V}$, we extract a watertight triangular surface via Marching Cubes at isovalue 0.5 (appropriate for binary occupancy), which creates vertices and faces by

[8] https://pmc.ncbi.nlm.nih.gov/articles/PMC3864549/; https://pmc.ncbi.nlm.nih.gov/articles/PMC3268159/

locally interpolating the isosurface inside each voxel cell.[9] As Marching Cubes is purely data-driven and local, it offers no control over the final vertex count. To reach a target complexity of 156 vertices, we therefore post-process the extracted mesh with quadric-error decimation, using a target *reduction ratio* rather than an exact vertex budget. Concretely, if the initial mesh has V_0 vertices, we set

$$\texttt{target\_reduction} \;=\; 1 - \frac{156}{V_0 + \delta},$$

with a small slack $\delta > 0$ (here, δ=10) to compensate for the well-known discreteness and topological constraints of edge/vertex contractions, which often cause practical under- or overshoot relative to the requested ratio. The decimator then attempts to remove approximately a fraction `target_reduction` of elements while minimizing geometric error in the quadric sense,[10] and typical implementations expose this control as a "fraction of the original mesh to remove" (rather than a guaranteed final count).[11] Because neither isosurface extraction nor decimation is deterministic with respect to an *exact* vertex number, the above strategy translates the desired budget (156) into a reduction ratio that empirically lands at, or very close to, the target after simplification, while preserving the mean shape encoded by the majority-vote voxel aggregation.

A.7.2 Constructing a Unit Sphere Prior with 156 vertices

To obtain a unit-radius reference sphere with a prescribed complexity, we start from an icosphere constructed by recursively subdividing a regular icosahedron. For a recursion level s, the number of faces grows as 4^s and the vertex count follows

$$V_0 \;=\; 12 + 10(4^s - 1) \;=\; 10 \cdot 4^s + 2,$$

so with `subdivisions=2` we get $V_0 = 162$ vertices on a unit sphere (`radius=1.0`). We then translate the desired budget of 156 vertices into a decimation ratio

$$\texttt{target\_reduction} \;=\; 1 \;-\; \frac{156}{V_0} \;=\; 1 - \frac{156}{162},$$

and apply quadric-error decimation to the `trimesh` icosphere via `fast_simplification.simplify_mesh` (wrapped through `pyvista`). In this workflow, the decimator interprets `target_reduction` as the *fraction of primitives to remove* and seeks a lowest-error approximation in the quadric sense, rather than enforcing a hard, exact vertex constraint. Consequently, the final mesh typically lands at (or extremely close to) the target $|\mathcal{V}| = 156$ on this highly regular input; if an exact count is required, one can perform a trivial post-check and adjust the reduction ratio by a negligible amount to reach 156 without materially altering geometry. This procedure leverages: (i) the closed-form vertex/face growth of recursively subdivided icosahedra to initialize near the target, and (ii) well-established quadric-error simplification to fine-tune the vertex budget while preserving the spherical shape.

[9] https://scikit-image.org/docs/0.23.x/auto_examples/edges/plot_marching_cubes.html; https://www.cs.toronto.edu/~jacobson/seminar/lorenson-and-cline-1987.pdf; https://thomas.lewiner.org/pdfs/marching_cubes_jgt.pdf

[10] https://www.cs.cmu.edu/~garland/Papers/quadrics.pdf

[11] https://docs.pyvista.org/api/core/_autosummary/pyvista.PolyDataFilters.decimate.html; https://pypi.org/project/fast-simplification/0.1.0/

A.8 Augmentation

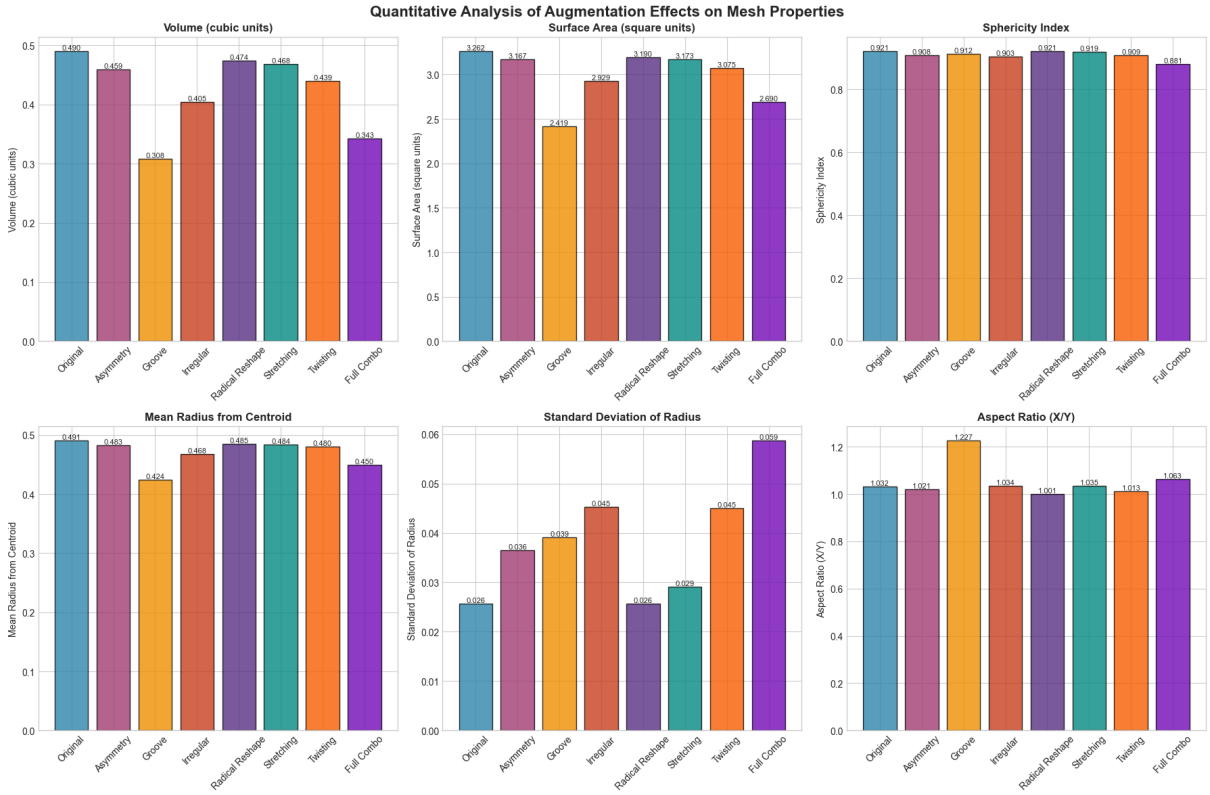


Figure A.33: Augmentation/Original

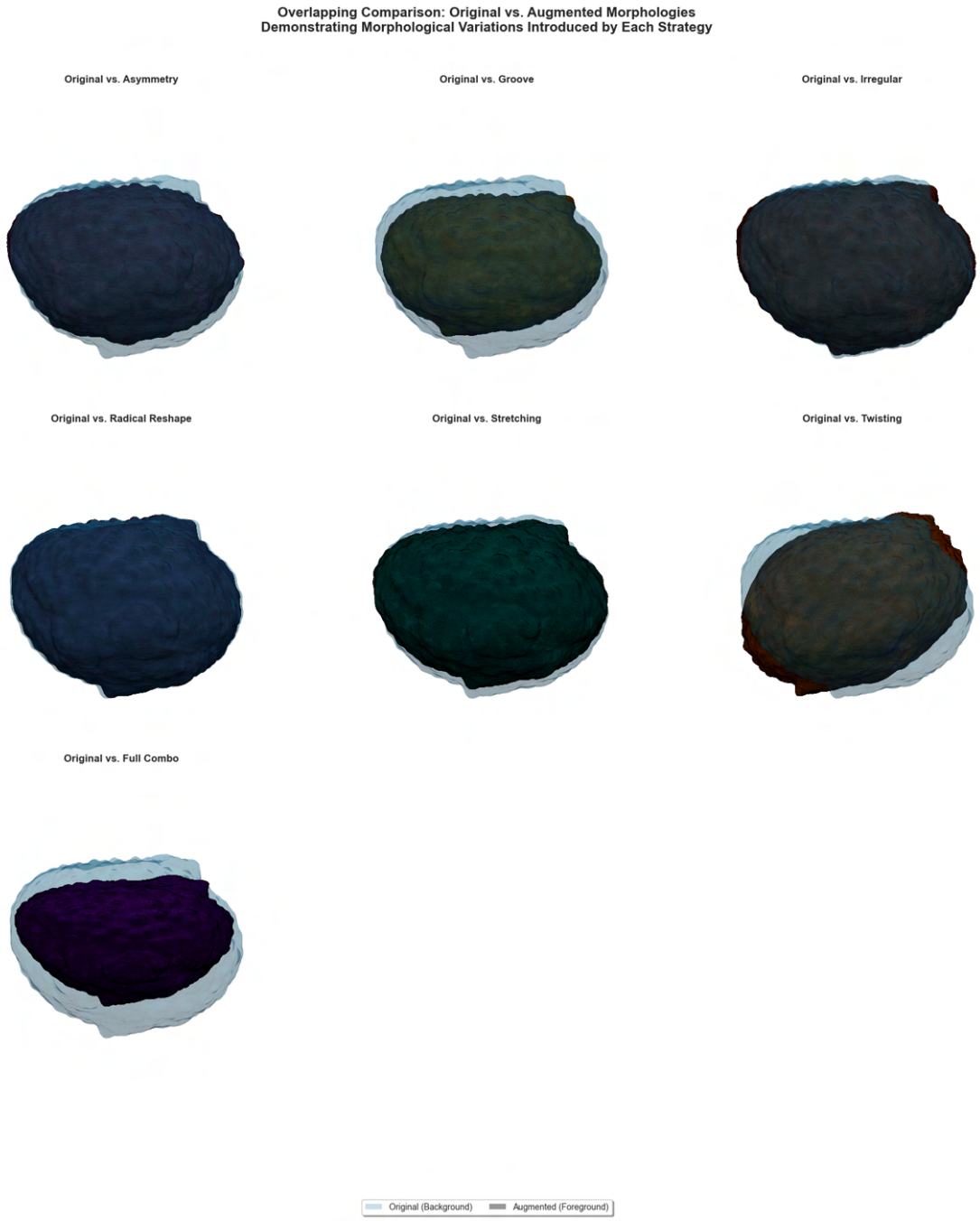


Figure A.34: Original overlapping with Augmented

A.8.1 Augmentation intensity

The augmentation intensity in this Blender-based mesh deformation system is based on a **level-based** progression model combined with **random-based** variation. The base intensity parameter t_i is calculated deterministically for each augmentation i and is subsequently modulated by stochastic factors. The calculation follows a linear scaling that ensures that early augmentations exhibit subtle deformations, whereas later augmentations exhibit progressively stronger changes. Each deformation method additionally applies random multipliers to the base intensity to achieve natural variation within each intensity level. This combination of deterministic progression and stochastic variation enables controlled and diverse mesh-augmentations.

$$t_i = \begin{cases} \dfrac{i}{N-1} \cdot 0.4 & \text{if } N > 1 \\ 0 & \text{if } N = 1 \end{cases}$$

$$D_{\text{method}}(t_i) = B_{\text{method}} \cdot (1 + t_i \cdot S_{\text{method}}) \cdot \mathcal{R}_{\text{method}}$$

$$\text{angle}_{\text{twist}} = (0.1 + t_i \cdot 0.4) \cdot \mathcal{U}(-1.2,\, 1.2)$$

Stretching:

$$\text{factor}_{\text{stretch}} = \frac{0.08 + t_i \cdot 0.35}{2.5} \cdot \mathcal{U}(0.85,\, 1.25)$$

$$\text{factor}_{\text{asym}} = (0.10 + t_i \cdot 0.30) \cdot \mathcal{U}(0.6,\, 1.6)$$

$$\text{angle}_{\text{groove}} = (-0.15 - t_i \cdot 0.3) \cdot \mathcal{U}(0.8,\, 1.2)$$

Where:

- $i \in \{0, 1, 2, \ldots, N-1\}$: augmentation index
- N: total number of augmentations (`num_augmentations`)
- B_{method}: base deformation strength for the respective method
- S_{method}: scaling factor for intensity-dependent amplification
- $\mathcal{U}(a, b)$: uniformly distributed random variable in interval $[a, b]$
- $\mathcal{R}_{\text{method}}$: method-specific random component

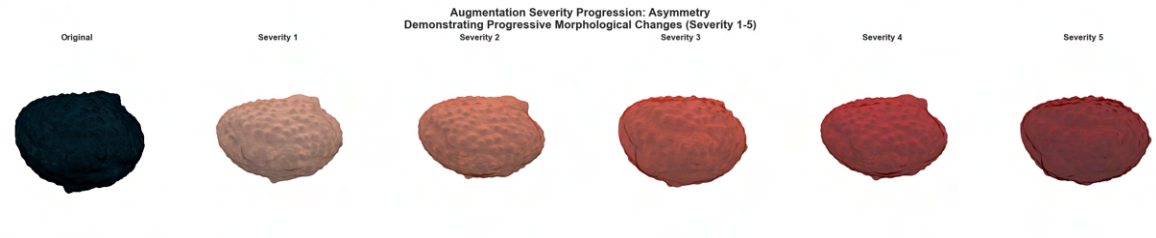


Figure A.35: Augmentation Severity Stage based with Randomness multiplier

A.8.2 Mathematical Properties of Manifoldness & Watertightness

To ensure that each pollen grain mesh $\mathcal{M}$ with vertices V and faces F becomes a valid 2-manifold and watertight surface, we require the following properties:

- **Edge-manifoldness**. Every (undirected) edge $e = (v_a, v_b)$ shared by vertices $v_a, v_b \in V$ must belong to exactly two distinct triangular faces $f, f' \in F$. Formally, letting

$$\deg_{\mathcal{M}}(e) \;=\; \big|\{\, f \in F : e \subset f \,\}\big|,$$

  we demand

$$\deg_{\mathcal{M}}(e) \;=\; 2, \quad \forall\, e \in E(\mathcal{M}),$$

  where $E(\mathcal{M})$ is the set of all edges in $\mathcal{M}$.
- **Vertex-manifoldness**. For each vertex $v \in V$, the union of all faces incident to v, denoted $\mathrm{star}(v)$, must form a single, topologically-connected fan (i.e., the faces around v should be edge-connected after removing v). Equivalently, there must be no "pinch" or "bowtie" configurations.
- **Non-self-intersection**. No two triangles in F may intersect in their interiors. That is, for any pair of distinct faces $f_i, f_j \in F$, their (closed) triangular areas satisfy

$$\big(\,\mathrm{int}(f_i)\big) \,\cap\, \big(\,\mathrm{int}(f_j)\big) \;=\; \varnothing.$$

- **Closedness (Watertightness)**. A mesh is watertight if it is already edge-manifold (every edge has two incident faces) and has no boundary edges. Formally, there must be no edge e with $\deg_{\mathcal{M}}(e) = 1$. Equivalently, the mesh must have no boundary loops:

$$\partial_{\mathcal{M}} \;=\; \varnothing,$$

  where $\partial_{\mathcal{M}} = \{\, e : \deg_{\mathcal{M}}(e) = 1 \,\}$.
- **2-Manifold Condition (Combined)**. Summarizing the above, $\mathcal{M}$ is a *2-manifold* if and only if:

$$\forall\, e \in E(\mathcal{M}) :\, \deg_{\mathcal{M}}(e) = 2,$$
$$\forall\, v \in V :\, \mathrm{star}(v) \text{ is connected and edge-manifold,}$$
$$\text{No two faces intersect except at shared edges or vertices.}$$

  When additionally $\mathcal{M}$ has zero boundary edges, it is *watertight*.
- **Poisson Reconstruction**. After removing interior faces, filling all boundary loops, and deleting faces adjacent to any non-manifold edge, we obtain an intermediate mesh $\overline{\mathcal{M}}$ satisfying

$$\deg_{\overline{\mathcal{M}}}(e) \le 2, \quad \forall\, e \in E(\overline{\mathcal{M}}).$$

  Applying Screened Poisson surface reconstruction, as described by [47], provides a new mesh $\widehat{\mathcal{M}} = (\widehat{V}, \widehat{F})$ extracted as the level-set

$$\widehat{\mathcal{S}} \;=\; \{\mathbf{x} \in \mathbb{R}^3 : \chi(\mathbf{x}) = 0\},$$

  where χ solves the screened Poisson equation

$$(\Delta + \alpha)\,\chi(\mathbf{x}) \;=\; \nabla\cdot\mathbf{V}(\mathbf{x}) \;+\; \alpha\,\chi_0(\mathbf{x}), \quad \mathbf{x} \in \mathbb{R}^3.$$

  By construction, $\widehat{\mathcal{M}}$ is now both watertight and 2-manifold.

A.9 Preprocessing of Holographic Images

Our models learned from synthetic pictures: a single gray pollen grain on a white background, centered and well-scaled. Real holographic images look different: they contain faint "ripples" and uneven background. We therefore apply a small, fixed sequence of image operations that makes real inputs look and behave like the synthetic ones, without inventing new details.

Each real sample consists of two synchronized images. We apply the same preprocessing to both views and feed them together to the 3D model.

What we do

Given a gray image I, we produce a cleaned image $\hat{I}$.

1. **Normalize and gently blur**. We first rescale intensities to $[0, 1]$ and apply a light Gaussian blur. This reduces tiny interference patterns while keeping the pollen outline visible.
2. **Find the pollen (thresholding)**. We convert the image to black/white with Otsu's global threshold [48]. Pixels darker than their reference level are marked as "pollen".
3. **Remove thin ripples**. We run a morphological opening with an elliptical kernel. We think of this as shaving off very thin, ring-like fringes while keeping the thicker pollen shape.
4. **Paint the background white**. Using the cleaned mask, we keep pollen pixels from the original and set everything else to pure white. If no pollen is found, we return a white image instead of guessing.
5. **Crop, scale and center**. We cut a tight box around the pollen, scale it up isotropically (same factor in both directions) but never above a safe cap $s_{\max}$, then paste it centered on a white canvas and resize to the target input size (e.g. 224×224).

If the mask is empty, we return a white frame; this avoids introducing artifacts. All operations are fast and work well in batches. This is however never the case for the samples we use in the relevant Experiment 5.4. We handpicked one sample pair per species from the provided Swisens Poleno dataset to make sure we always have some sort of holographic scan visible.

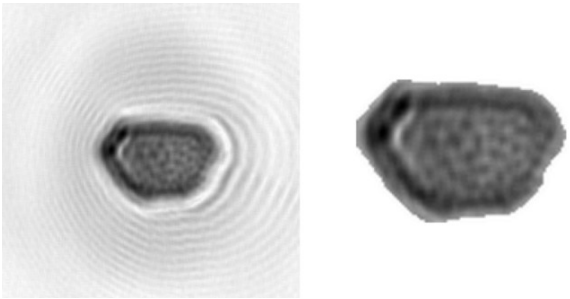


Figure A.36: Holographic scan (left) vs. the same pollen grain after our preprocessing steps were applied (right).

Why these choices help

Each step addresses one gap between real and synthetic data:

- Blur reduces holographic ripples and turns them to a small range of gray background colors that can be cut out via thresholding.
- Threshold separates pollen from background with a global rule (Otsu) when illumination is uneven.
- Morphology removes thin artifacts but keeps the main body.

- White background & centering matches the synthetic look, we trained our models on with the 3D Pollen Library dataset, and removes nuisance variation in position and scale.