# Appendix

## A. Data Collection and Processing Details

Figure I shows a selection of our data collection locations. Different colors incidate different location types.
Figure II shows more visualization on our 3D reconstruction results.

## B. 3DGS Training Details

**Training Setup**. Our 3DGS implementation is built on top of the open-source `gsplat` [93] framework, which provides an efficient and scalable renderer for Gaussian splatting. For all experiments, we render and train at a fixed resolution of 800×800 pixels. This resolution offers a good balance between spatial detail and GPU memory usage, and allows us to handle large urban scenes without exhausting GPU memory. Other hyperparameters for 3DGS training is listed in Tab. I.

**Initialization from metric point clouds**. For each scene, we initialize 3D Gaussians from the dense colorized point cloud produced by MetaCam Studio. The raw point cloud has a spacing of 5–10 mm, resulting in roughly 10–50 million points per scene. To keep training tractable while preserving sufficient detail for a five-minute walking-scale street scene, we uniformly downsample the point cloud to around 5 million points per scene and create one Gaussian per point. Our initialization largely follows the default settings in `gsplat`: we use a k-nearest-neighbor heuristic to set the initial *scale* of each Gaussian, and parameterize initial opacity inversely proportional to initial volume. This density-based parameterization prevents large Gaussians or spurious floaters left by transient obstacles from artificially dominating the rendering at the beginning of training.

**Depth Regularization**. Because the global point cloud is extremely dense and globally consistent, the resulting depth maps rendered from initialized Gaussians provide a close approximation to ground-truth geometry. This term acts as a geometric prior: it regularizes Gaussians along the viewing rays and prevents them from drifting into free space or collapsing towards the cameras during optimization.

A natural alternative, enabled by accurate LiDAR point cloud, is to freeze the Gaussian means and only optimize appearance-related parameters such as color, opacity, and rotation. We experimented with this stronger form of supervision and found that, although it indeed preserves excellent multi-view geometric consistency, it yields suboptimal visual quality and can still produce degenerate behavior around training views, as shown in Tab. II. In contrast, relying purely on image supervision from a limited set of views improves per-view fidelity but tends to sacrifice consistency for unseen viewpoints. Our depth regularization strikes a balance between these extremes: it keeps the learned geometry close to the dense metric point cloud while still allowing Gaussians to move and adapt to fit high-quality images.


Figure I. **Data collection locations**. We collect data in New York City and Jersey City. The capture location is carefully selected to cover different scenes.

Table I. Hyperparameters for our 3DGS training.

| Hyperparameter | Value |
|---|---|
| Camera model | Pinhole |
| Camera FOV | 120° |
| Image resolution | 800×800 |
| Training steps | 15,000 |
| SH degree | 3 |
| Initial opacity | 0.99 |
| Initial scale | 0.5 |
| Perceptual loss weight | 0.2 |
| Depth loss weight | 0.02 |
| Means learning rate | $1.6 \times 10^{-5}$ |
| Scales learning rate | $1.0 \times 10^{-3}$ |
| Opacity learning rate | $2.0 \times 10^{-2}$ |
| Quaternion learning rate | $1.0 \times 10^{-3}$ |
| SH band 0 learning rate | $5.0 \times 10^{-4}$ |
| SH band N learning rate | $1.25 \times 10^{-4}$ |
| Opacity regularization | 0 |
| Scale regularization | 0.01 |

Table II. **Comparison of geometric priors**. We compare our depth-based regularization with the alternative strategy of freezing Gaussian centers to the LiDAR point cloud.

| Geometric Prior | Interpolated Views | | | Extrapolated Views | | |
|---|---|---|---|---|---|---|
| | PSNR ↑ | SSIM ↑ | LPIPS ↓ | PSNR ↑ | SSIM ↑ | LPIPS ↓ |
| Depth Loss | 20.37 | 0.688 | 0.327 | 17.92 | 0.591 | 0.445 |
| Frozen Gaussians | 20.39 | 0.703 | 0.327 | 17.10 | 0.558 | 0.456 |

**Training view augmentation**. We utilize the pretrained Difix3D+ [77] model to augment our training camera views. Instead of modifying any original dataset images, we follow a self-training strategy inspired by Difix3D+: for a sampled extrapolated camera pose, we first rasterize an image from the current 3DGS model, then feed it together with its nearest neighboring training views as references into Difix3D+

Figure II. **Our 3D reconstruction from LIV-SLAM**. The global colorized point cloud is downsampled for visualization cleanness.

Table III. **Ablation study on 3DGS training components**. "P" stands for pinhole and "F" stands for fisheye.

| 3DGS Components | | | | | | Interpolated Views | | | Extrapolated Views | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| ***Diverse Views*** | ***GT Camer Pose*** | ***LiDAR Point Cloud*** | ***Depth Loss*** | ***View Augmentation*** | ***Camera Model*** | **PSNR ↑** | **SSIM ↑** | **LPIPS ↓** | **PSNR ↑** | **SSIM ↑** | **LPIPS ↓** |
| | | | | | P | 15.26 | 0.515 | 0.558 | 12.99 | 0.457 | 0.655 |
| ✓ | | | | | P | 18.27 | 0.658 | 0.510 | 16.90 | 0.624 | 0.559 |
| ✓ | ✓ | | | | P | 15.14 | 0.565 | 0.693 | 14.66 | 0.546 | 0.716 |
| ✓ | ✓ | ✓ | | | P | 15.54 | 0.532 | 0.605 | 15.19 | 0.515 | 0.625 |
| ✓ | ✓ | ✓ | ✓ | | F | 20.91 | 0.681 | 0.262 | 16.87 | 0.530 | 0.468 |
| ✓ | ✓ | ✓ | ✓ | | P | 20.95 | 0.696 | 0.269 | 16.97 | 0.544 | 0.452 |
| ✓ | ✓ | ✓ | ✓ | ✓ | P | 20.37 | 0.688 | 0.327 | 17.92 | 0.591 | 0.445 |

to obtain a cleaned and geometrically accurate novel view. The synthesized image is then added back into the training set and supervised with a lower loss weight, so that it doesn't the original observations.

Our augmentation strategy is different from that in Difix3D+, which mainly densifies views along specific paths to improve rendering quality near interpolated evaluation trajectories. We adopt a more general sampling strategy tailored to large-scale navigation scenes. At the early stages of training, we only sample and refine novel views in a small neighborhood around the training trajectories. Along the training steps, we gradually increase the sampling radius and move the extrapolated viewpoints farther away, effectively implementing a curriculum over viewpoint distance. This procedure increases the diversity and coverage of training views in a controlled manner, which is crucial for stabilizing large-scale 3DGS training and improving generalization to truly unseen viewpoints.

**Ablation study**. Table III studies different 3DGS training design choices. We observe that while ground truth camera pose and dense LiDAR points provides a good initialization, they have to be well utilized by a good regularization signal (*e.g.* the depth loss) to get good view synthesis quality. Moreover, while training view augmentation doesn't neces-

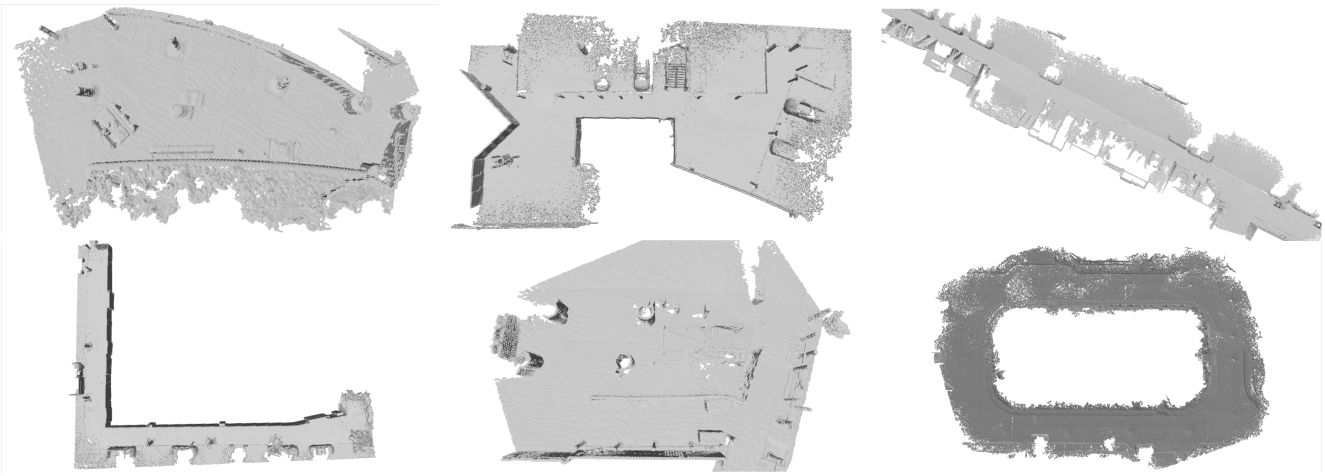

Figure III. **Our mesh extraction from global point cloud**. The mesh is cut by a height threshold and a radius threshold around capturing cameras. This ensures a lightweight mesh while keeping geometry consistency.

sarily provide better interpolated view synthesis quality, it shows significantly better results on extrapolated views.

## C. Mesh Extraction & Scene Integration

Since our primary goal is to support navigation and collision checking rather than fine-grained physical interaction, we design the mesh representation to be occupancy-accurate and efficient rather than visually detailed. Starting from the global metric point cloud, we first filter out points that are too far from the sensor trajectories, as these regions are unlikely to be reachable or relevant for agent interaction. We then voxelize the remaining points into a 3D occupancy grid and run a standard Marching Cubes [78] algorithm to extract a triangle mesh. This occupancy-based reconstruction provides a controllable trade-off between resolution and complexity while preserving the spatial support of walkable surfaces and major obstacles. After meshing, we perform a lightweight cleaning step to remove residual noise and irrelevant details. We discard small isolated components (*e.g.*, mesh fragments with fewer than 50 faces), which typically correspond to transient objects or reconstruction artifacts. The resulting mesh (Fig. III) does not need to be strictly watertight, as visual appearance is handled by the 3DGS model.

We integrate the learned 3DGS model and the collision mesh into a single USD scene, using the MetaCam world coordinate frame (in meters) as the common reference. In this representation, the mesh provides a lightweight physics and collision layer, while the 3DGS model is used as the primary renderer. The resulting USD scenes can be directly loaded into simulators such as Isaac Sim, and the same collision mesh can be imported into Unity for baking navigation meshes as described below.

## D. Expert Trajectory and VLN

**Expert trajectory**. We import the collision mesh into Unity and use its built-in NavMesh baking API to extract a triangulated navigable surface. Given any pair of start and goal locations, we align them onto the closest points on the NavMesh and invoke Unity's pathfinding module to compute a collision-free expert trajectory on this surface. To ensure that the paths are semantically meaningful and well grounded in the captured data, start and goal candidates are sampled in the vicinity of camera poses from the training or test splits, so that each endpoint corresponds to a visually interpretable location in the scene.

**Language instruction**. Once an expert trajectory is obtained, we replay it in our 3DGS-based simulator to render an egocentric video along the path. We then feed both the rendered video and the corresponding trajectory summary to the Gemini 2.5 Flash model [84] to automatically generate natural-language instructions. To improve controllability and consistency, we adopt a two-stage prompting scheme: the model is first asked to output a structured JSON description that segments the route into sub-instructions with associated landmarks and actions, and is then prompted to condense this JSON representation into a single fluent instruction. Compared to fully manual annotation, this procedure is substantially more scalable and yields more stable instructions across scenes, while still allowing for quality control: we perform manual spot checks and discard the very small fraction of instructions that are clearly inconsistent or unusable. The resulting set of expert trajectories and instructions is stored together with the underlying 3DGS scenes, making Wanderland a unified testbed for different embodied navigation tasks.

# E. More Experiment Details

## E.1. 3D Reconstruction

**Evaluation metrics**. We use different evaluation metrics to assess different aspects in camera pose estimation accuracy. The definitions are detailed below:

- Absolute Trajectory Error (ATE) measures the global consistency between predicted and ground truth trajectories after alignment.
- Translation ATE - Raw (**T-ATE$^{\mathbf{R}}$**): Root mean square error (RMSE) of translation after SE(3) alignment:

$$\text{T-ATE}^{\text{R}} = \sqrt{\frac{1}{N}\sum_{i=1}^{N} \|(R_{SE3} \cdot \mathbf{t}_i^{pred} + \mathbf{t}_{SE3}) - \mathbf{t}_i^{gt}\|^2}, \tag{1}$$

  where $R_{SE3}$ and $\mathbf{t}_{SE3}$ are the rotation and translation from SE(3) alignment.
- Translation ATE - Scaled (**T-ATE$^{\mathbf{S}}$**): RMSE of translation after SIM(3) alignment. This metric evaluates relative-scale pose accuracy independent of absolute scale:

$$\text{T-ATE}^{\text{S}} = \sqrt{\frac{1}{N}\sum_{i=1}^{N} \left\|s \cdot R_{\text{SIM3}}\mathbf{t}_i^{\text{pred}} + \mathbf{t}_{\text{SIM3}} - \mathbf{t}_i^{\text{gt}}\right\|^2}, \tag{2}$$

  where $R_{SIM3}$, $\mathbf{t}_{SIM3}$, and $s$ are the rotation, translation, and scale from SIM(3) alignment.
- Rotation ATE (**R-ATE**): RMSE of rotation angles:

$$\begin{aligned} \text{R-ATE} &= \sqrt{\frac{1}{N}\sum_{i=1}^{N} \Delta\theta_i^2}, \\ \Delta\theta_i &= \cos^{-1}\left(\frac{\text{tr}(\mathbf{R}_i^{gt\top}\mathbf{R}_i^{pred}) - 1}{2}\right), \end{aligned} \tag{3}$$

  where $\mathbf{R}_i$ denotes the rotation matrix.
- Relative Trajectory Error (RTE) measures the consistency of relative motions between camera pairs.
- Translation RTE (**T-RTE**): RMSE of relative translation distances between all camera pairs:

$$\begin{aligned} \text{T-RTE} &= \sqrt{\frac{2}{N(N-1)}\sum_{i=1}^{N-1}\sum_{j=i+1}^{N} (\Delta_{ij})^2}, \\ \Delta_{ij} &= \|\mathbf{t}_i^{pred} - \mathbf{t}_j^{pred}\| - \|\mathbf{t}_i^{gt} - \mathbf{t}_j^{gt}\|. \end{aligned} \tag{4}$$

- **T-RTE (degrees)**: RMSE of angular differences in relative translation directions. Similarly to T-ATE$^{\text{S}}$, it also evaluates relative-scale pose accuracy independent of absolute scale:

$$\begin{aligned} \text{T-RTE (deg)} &= \sqrt{\frac{2}{N(N-1)}\sum_{i=1}^{N-1}\sum_{j=i+1}^{N} \theta_{ij}^2}, \\ \theta_{ij} &= \cos^{-1}\left(\frac{(\mathbf{t}_i^{pred} - \mathbf{t}_j^{pred}) \cdot (\mathbf{t}_i^{gt} - \mathbf{t}_j^{gt})}{\|\mathbf{t}_i^{pred} - \mathbf{t}_j^{pred}\|\|\mathbf{t}_i^{gt} - \mathbf{t}_j^{gt}\|}\right). \end{aligned} \tag{5}$$

- Rotation RTE (**R-RTE**): RMSE of relative rotation angles between all camera pairs:

$$\begin{aligned} \text{R-RTE} &= \sqrt{\frac{2}{N(N-1)}\sum_{i=1}^{N-1}\sum_{j=i+1}^{N} \Delta\theta_{ij}^2}, \\ \Delta\theta_{ij} &= \cos^{-1}\left(\frac{\text{tr}(\mathbf{R}_{ij}^{gt\top}\mathbf{R}_{ij}^{pred}) - 1}{2}\right), \end{aligned} \tag{6}$$

  where $\mathbf{R}_{ij} = \mathbf{R}_i\mathbf{R}_j^\top$ represents the relative rotation.
- **AUC@30:** Area under the curve of the cumulative distribution of maximum relative errors, with a maximum threshold of 30 degrees:

$$\text{AUC@30} = \int_0^{30} P(\max(\text{R-RTE}, \text{T-RTE}_{\text{deg}}) < \theta)d\theta. \tag{7}$$

  This provides a comprehensive measure of reconstruction quality across different error tolerances.
- Success Rate (**SR**) is defined to be the ratio of scenes with AUC@30 $> 0.1$.

**Evaluation setup**. Due to GPU memory limitation, some reconstruction models (DUSt3R and VGGT) can't process all images in a scene. To ensure fair comparison, we uniformly downsample all scenes to be below 500 images. This is another limitation of these methods.

## E.2. Photorealistic Sensor Simulation

**Evaluation metrics**. We use common metrics for evaluating novel view synthesis models:

- Peak Signal-to-Noise Ratio (**PSNR**): Measures pixel-wise reconstruction quality:

$$\text{PSNR} = 10\log_{10}\left(\frac{1}{\text{MSE}}\right), \tag{8}$$

  where $\text{MSE} = \frac{1}{WH}\sum_{i,j}(I_{i,j}^{pred} - I_{i,j}^{gt})^2$.
- Structural Similarity Index (**SSIM**): Evaluates structural preservation:

$$\text{SSIM} = \frac{(2\mu_x\mu_y + C_1)(2\sigma_{xy} + C_2)}{(\mu_x^2 + \mu_y^2 + C_1)(\sigma_x^2 + \sigma_y^2 + C_2)}, \tag{9}$$

  where $\mu, \sigma$ are local statistics.

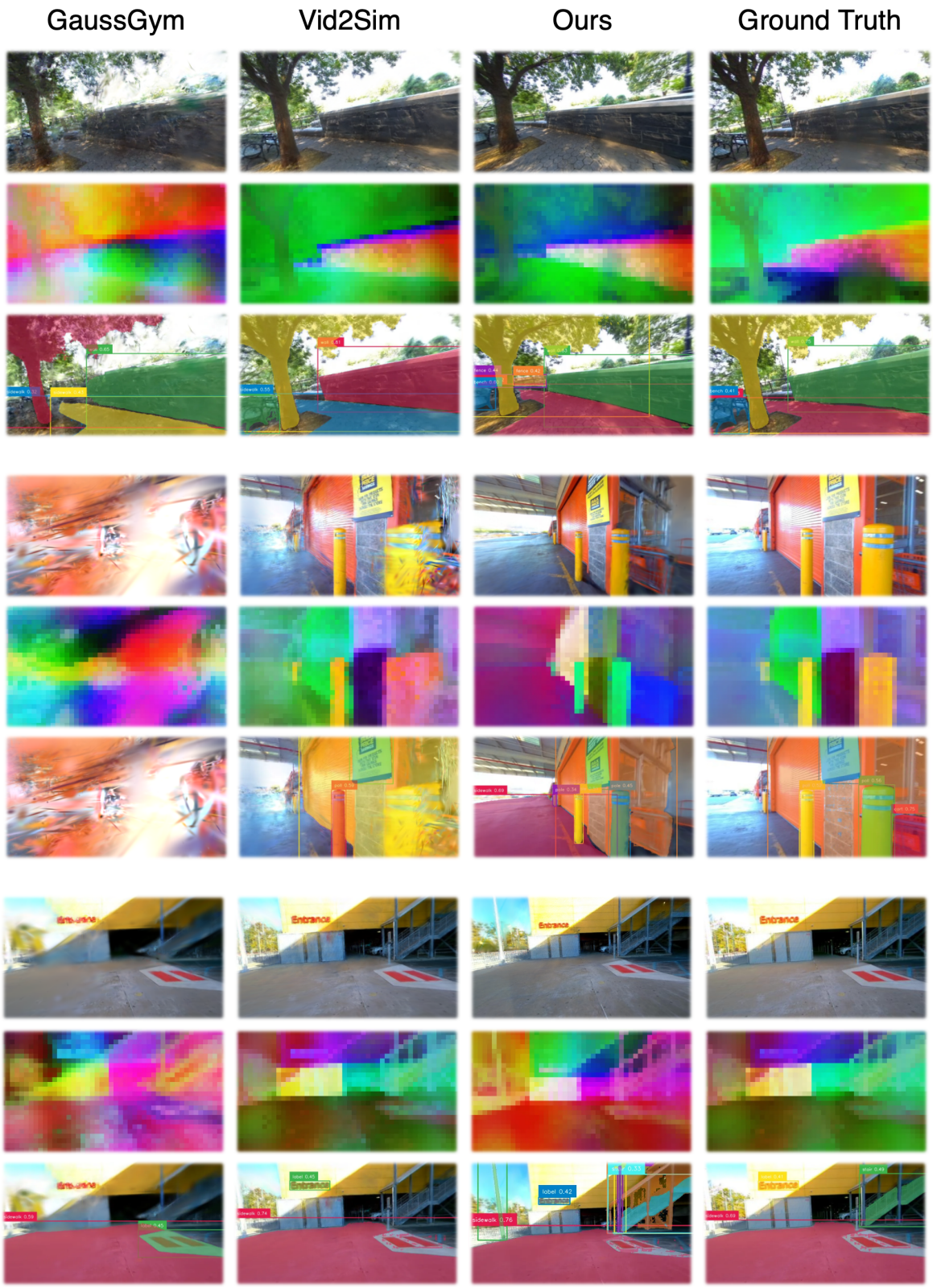


Figure IV. **More visualization on photorealistic and semantic consistent sensor simulation**. Format is the same as Fig. 6.

- Learned Perceptual Image Patch Similarity (**LPIPS**): Measures perceptual quality using deep features:

$$\text{LPIPS} = \sum_{l} \frac{1}{H_l W_l} \sum_{h,w} \|w_l \odot (\phi_l(I^{pred})_{h,w} - \phi_l(I^{gt})_{h,w})\|_2^2, \tag{10}$$

where $\phi_l$ is the deep features from the VGGNet [94].

**More results**. Figure IV shows more qualitative comparisons on different sim-to-real pipelines.

### E.3. Embodied Navigation

**Evaluation metrics**. We use three common metrics and a self-designed metric to evaluate embodied navigation tasks:

- Navigation Error (**NE**): Euclidean distance between the agent's final position and the goal location upon episode termination.
- Success Rate (**SR**): Percentage of episodes where the agent successfully reaches the goal within the specified maximum steps.
- Success weighted by Path Length (**SPL**): Combined metric considering both success and path efficiency:

$$\text{SPL} = \frac{1}{N} \sum_{i=1}^{N} S_i \frac{l_i}{\max(p_i, l_i)}, \tag{11}$$

where $S_i$ is success (0/1), $l_i$ is optimal path length, and $p_i$ is actual path length for episode $i$.

- Intervention Rate (**IR**): Percentage of episodes requiring early termination due to non-timeout failures including agent becoming stuck, making no progress, or exiting scene boundaries.

**RL training setup.** For the "RL post-trained" rows in our experiments, we fine-tune the released navigation policies of NoMaD, CityWalker, and MBRA on Wanderland using a standard on-policy reinforcement learning setup. All agents interact with our 3DGS-based simulator described in the main paper and operate in the same action space as in their original implementations. Episodes are sampled from the training split by drawing start and goal locations on the navigable NavMesh as in Sec. 3, and each episode is capped at a fixed maximum number of steps (1000 in our experiments). Episodes may also terminate early when the agent reaches the goal, gets stuck, or leaves the valid navigation region. We use Proximal Policy Optimization (PPO) with generalized advantage estimation (GAE), a $\gamma = 0.99$ discount factor. Unless otherwise noted, we only update the policy, so that RL mainly adapts high-level navigation behavior to the geometry and semantics of Wanderland.

**Reward design.** The reward function follows a simple distance-based shaping scheme with explicit penalties for unsafe behaviors. Let $d_t$ denote the distance from the agent to the goal at time step $t$. The per-step reward $r_t$ is defined as

$$r_t = \begin{cases} +R_{\text{succ}}, & \text{if the agent reaches the goal,} \\ -R_{\text{fail}}, & \text{if the episode early terminated,} \\ -\alpha + \beta\,(d_{t-1} - d_t), & \text{otherwise,} \end{cases} \tag{12}$$

where $R_{\text{fail}} > 0$ is a terminal success bonus, $R_{\text{fail}} > 0$ controls the penalty for unsafe terminations, $\alpha > 0$ is a small step penalty that encourages shorter paths, and $\beta > 0$ weights the progress reward given by the reduction in geodesic distance. In words, the agent is rewarded for making progress toward the goal, slightly penalized for every time step, strongly rewarded upon success, and explicitly penalized when it falls, gets stuck, or ignores obstacles and exits the valid navigation area. This shaping aligns the optimization objective with our evaluation metrics: minimizing NE by rewarding geodesic progress, maximizing SR by giving success bonuses, improving SPL by discouraging unnecessarily long paths, and reducing IR by penalizing unsafe behaviors that trigger early termination. We use a single set of reward coefficients across all methods to ensure a fair comparison of RL post-training effects on Wanderland.