

Appendix

A. Vid2Sim dataset

Our Vid2Sim dataset includes 30 high-quality real-to-sim simulation environments. These environments are reconstructed from video clips sourced from 9 web videos recorded by individuals walking along streets with a single hand-held camera. Each clip capture includes 15 seconds of forward-facing video recorded at 30 fps, providing 450 frames per scene for reconstruction. To ensure privacy, we mask all human faces and identifiable information in the video frames.

A.1. Camera Reconstruction

Commonly used Structure-from-Motion (SfM) methods, such as COLMAP [47], often fail to reconstruct accurate camera poses from in-the-wild videos due to significant variations in viewpoints and lighting conditions. Instead, we employ GLOMAP [42], an advanced general-purpose SfM system that is more robust and efficient than COLMAP, to obtain accurate camera poses. We further compared its results with other learning-based methods [27, 57] in Tab. 4 to show its reliability.

A.2. Scene Statistics

We analyze our Vid2Sim dataset to demonstrate its diversity and statistics. The collected 30 videos contain diverse types of urban spaces (plazas: 5, sidewalks: 13, parks: 3, crosswalks: 3, residential streets: 6), diverse weather and lightning conditions (sunny: 20, cloudy: 7, rainy: 2, indoor lightning: 1), and diverse locations around the world (U.S.: 4, U.K.: 3, Finland: 20, Lithuania: 3). Our Vid2Sim dataset was designed to highlight the potential for training and evaluating the mobile robot in diverse real-world environments.

In Fig. 8, We show a preview snap-shot of the 30 diverse environments in our dataset, these environments encompass a variety of urban navigation scenarios designed for more robust and generalizable navigation policy training in complex urban environments.

B. Geometry-Consistent Reconstruction

B.1. Implementation details

In this section, we mainly introduce the implementation details of our geometry-consistent reconstruction method, including training strategy, hyper-parameter settings, and other extra details.

To avoid the influence of dynamic objects within the original videos, for each frame, we generate a dynamic mask to mask out all the moving objects within the scene with an off-the-shelf video-based tracker DEVA [10]. We

Methods	RPE (Translation) ↓	RPE (Rotation) ↓
DUST3R [57]	1.632	1.357
MASt3R [27]	1.622	1.160
COLMAP [47]	✗ (Failed)	✗ (Failed)
Ours (GLOMAP [†] [42])	1.611	1.057

Table 4. SfM evaluation on SiT [2] dataset. We report the relative pose error of the translation and rotation on 5 video sequences of street scenes with daily dynamic objects. Each sequence consists of a total of 200 frames. [†] indicates dynamic masks are applied

train our model for 30k iterations. We use the same training hyper-parameters as 3DGS [25] and adopt the densification strategy from AbsGS [69]. All the depth, normal, and geometry-consistency loss are started to apply at the 500 iterations still the end of the training.

For the mesh extraction, we first render the depth map from each frame and fuse them into a TSDF field with KinectFusion [21], the mesh is then extracted from TSDF field with voxel size set to 0.1. To eliminate the potential dynamics issue caused by uneven ground reconstructions, we further remove the ground plane of the scene in our mesh extraction through ground plane segmentation. We found that directly applying a general segmentation model (such as SAM [26] or SAM-HQ [24]) cannot remove all the ground mesh due to inaccurate segmentation and prompt ambiguity. Instead, we propose to generate a detailed ground mask \mathcal{M}_i by using the ground plane’s normal direction as a prior:

$$\mathcal{M}_i = \|\arccos(\mathbf{N}_i \cdot \bar{\mathbf{n}}'_i) < \delta\|, \quad (8)$$

where \mathbf{N}_i indicates the rendered normal map of the i^{th} frame, $\bar{\mathbf{n}}'_i$ denotes the mean normal direction of the ground surface computed from the ground segmentation mask given by the SAM-HQ [24] model and δ denotes the angle threshold. Pixels whose normal directions are within δ degrees of the mean ground surface normal direction are grouped together to generate the final ground mask. Incorporating normal priors results in a more precise ground mask, enabling clean mesh extraction without the ground surface. We set $\delta = 15$ degrees in our implementation.

B.2. Screen-space covariance culling

In this section, we systematically evaluate our screen-space covariance culling method both qualitatively and quantitatively. Results demonstrate this technique could effectively remove rendering artifacts and significantly improve agent observation quality.

Since floater artifacts often appear when the viewing angles and focal length differ significantly from the training

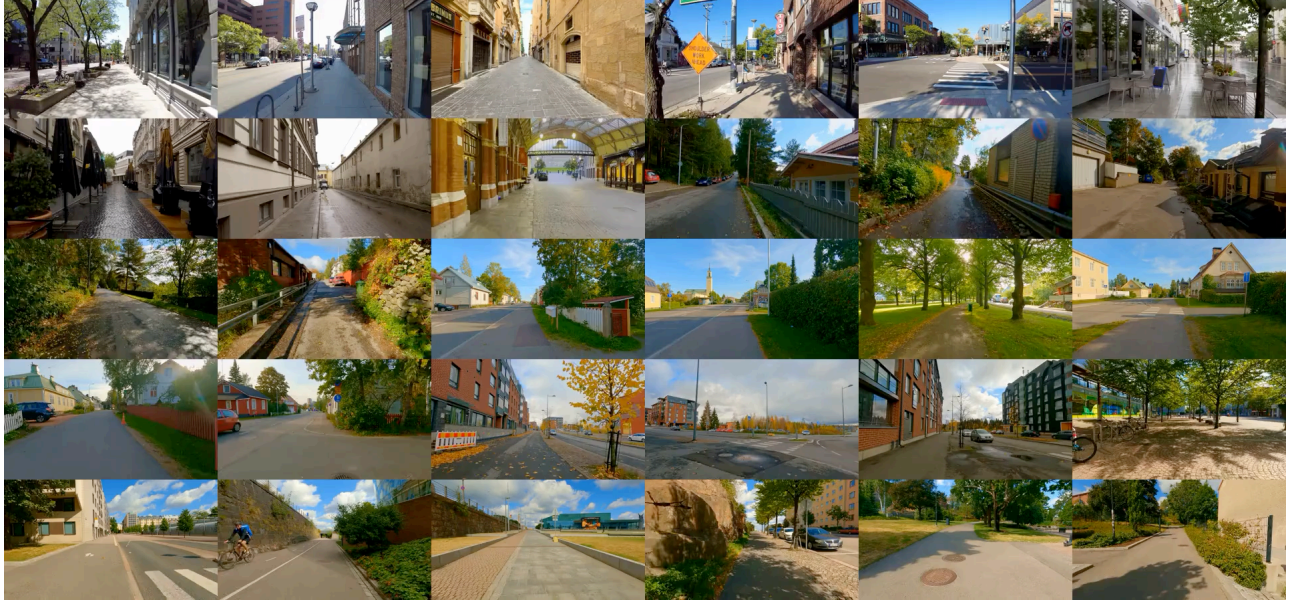


Figure 8. Vid2Sim Dataset Preview



(a) Before 2D Covariance Culling (b) After 2D Covariance Culling

Figure 9. Screen-space Covariance Culling Comparison

view. Therefore, standard test views that closely align with the training view may not accurately represent real situations during agent training. For better evaluation, we simulate the agent’s camera view by adjusting the test view’s camera focal down by 1.5x and shifting the camera down by 1 unit. Given the fact that there is no ground-truth image available to apply common NVS evaluation metrics (e.g. PSNR and SSIM) to evaluate the quality of the culled images, we instead provide a quantitative evaluation of screen-space covariance culling by computing the Fréchet Inception Distance (FID) [18] score between the training views and the agent’s rendered observations. The qualitative results show a substantial 10.7% improvement in the FID score after our screen-space covariance culling, which supports its effectiveness.

It is important to note that, due to the limited size of the evaluation dataset and the significant difference between the agent’s views and the training views, the absolute FID values cannot be directly compared to those commonly reported in generative models evaluation [19, 49]. Instead, the FID score in this context serves as a relative metric for us to better understand the improvements achieved through this covariance culling process.

Metric	w/o Culling	w Culling	Improvement (%)
FID [18]	214.47	191.54	+10.70%

Table 5. Comparison of FID scores for rendered images with and without 2D covariance culling.

C. Simulation Environment Setup

Our simulation environment is built based on the Unity [17] physics engine. Our agents, hybrid scene representation, and static and dynamic obstacles are imported together to form our diverse simulation environments. The ground surface is configured as a horizontal walkable area, while other scene meshes and obstacles are tagged as collidable objects. To maintain visual consistency, the ground and scene meshes are rendered invisible and serve only for collision interactions.

For the agent settings, we position the agent’s camera sensors at the front of the robot, matching the real-world sensor placement. The robot size in the simulation is scaled to metric scale with the background scene adjusted proportionally. The agent operates using a bicycle dynamics model, configured with a wheelbase of 0.8 meters and a maximum turning angle of 30 degrees. We introduce random perturbations to the camera’s position and rotation to simulate real-world disturbances caused by uneven ground during navigation.

In Fig. 10, We also provide an example of a digital twin environment created using our Vid2Sim pipeline that transforms a real-world environment into a simulation environment with the minimal sim-to-real gap. The agent RGB ob-

servation is provided at the top-left corner which replicates real-world observation.

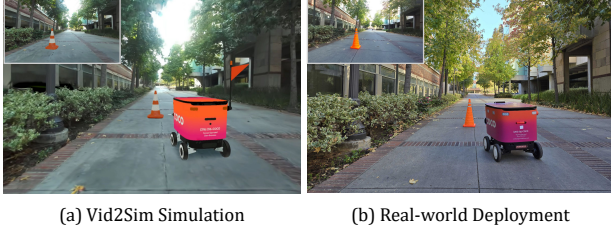


Figure 10. Digital-twin environment for real-world scene

D. RL Training Details

In this section, we provide the training details of our RL agents. To enable agent training with popular RL frameworks like OpenAI Gym [7] and Stable-Baselines3 [44], we compiled our Unity environments and integrated them with an environment wrapper to make them compatible with the OpenAI Gym interface. The simulation system is running at 50hz and the RL training is running at 5hz.

During each episode, the agents are randomly initialized from a starting point and need to navigate to another random goal point within the scene. Typically, the distance between the starting point and the goal point is around 10~30m. For both the PointNav and SocialNav tasks, an agent is considered successful if it reaches the goal within 0.5m.

D.1. Reward shaping

Our reward function for the agent is defined as follows:

$$R = R_{term} + c_1 R_{dist} + c_2 R_{steer} + c_3 R_{crash} + c_4 R_{time}$$

- Terminal reward R_{term} : is a sparse reward set to +10 if the agent successfully reaches the destination and -10 if it fails.
- Distance reward R_{dist} : is a dense reward defined as $R_{dist} = d_t - d_{t-1}$, where d_t represents the current distance between the agent to the goal point, which guide agent navigates towards the goal during training. We set the weight $c_1 = 1$.
- Steering smoothness reward R_{steer} : is a regularization reward defined as $R_{steer} = -\|s_t - s_{t-1}\| \cdot v_t$ to penalize inconsistent steer movement during agent navigation. The weight c_2 is set to 0.05
- Crash reward R_{crash} is used to penalize the collision between the agent and other objects, including the environment, static obstacles, and dynamic agents. It's a dense negative reward defined as $-1(c_t)$ where c_t denotes the collision happens at time t and $1(\cdot)$ is the indicator function. We set the weight of R_{crash} as $c_3 = 1.0$.
- Time reward R_{time} is a dense reward defined as $R_{time} = -\Delta t$, between two time steps the R_{time} is simplifies to -1 to encourage efficient navigation by penalizing longer episodes. We set the weight c_4 of R_{time} to -0.1

At any time step t if the $R_{term} \neq 0$, the episode will terminate. The agent is considered as failed and receives a $R_{term} = -10$ under the following conditions: 1) Navigating outside the drivable area. 2) Exceeding 3000 time steps in the episode, resulting in a timeout. 3) Accumulating more than 3 collisions within an episode. During the testing, any collision between the agent and other objects will result in a cost +1.

D.2. Hyper-parameter settings

We train our agent with 30 parallel environments and the training takes around 15 hours on a single NVIDIA A5000 GPU. We provide our hyper-parameter settings in Tab. 6

SAC Hyper-parameters	Value
Learning starts	10000
τ (Target critic update ratio)	0.005
Discount factor γ	0.99
SDE sample frequency	64
Batch size	256
Learning rate	3×10^{-4}
Use SDE at warmup	True
Use SDE	True

Table 6. SAC Hyper-parameters used in experiments.

E. Sim-to-real Deployment

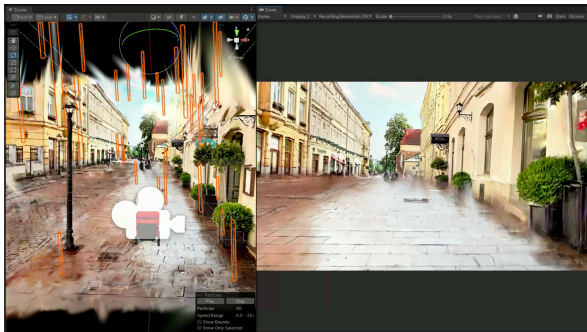
For the real-world experiment, we deploy navigation policies trained in Vid2Sim on the same four-wheeled delivery robot used in the simulation. The robot takes RGB images as inputs directly from an onboard camera, which has the same resolution of 1280×720 and the same intrinsic and extrinsic parameters as the sensor specifications in simulation. We resize the current image to 128×72 and stack it with the images from the past 5 timesteps to incorporate the historical information. We then combine the image inputs and the distance and heading angle to the goal point from robot odometry as the policy observation. The action output includes the normalized linear and angular velocities between -1 and 1, and we perform system identification to align the dynamics of the real robot with the simulation by re-scaling the normalized velocities in real-world units, and we use the built-in controller of the robot to convert the velocity commands to the low-level motor controls for actions.

F. Particle Simulation for Weather Editing

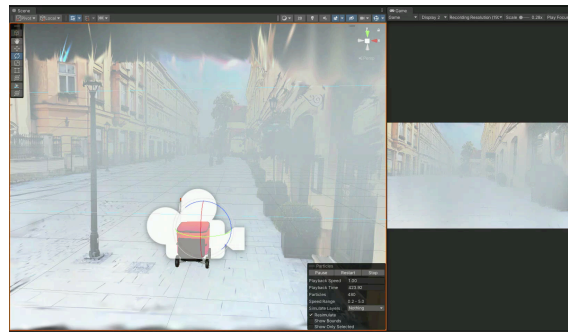
Apart from global scene layout editing illustrated in the main paper, we also demonstrate our ability to simulate different weather conditions like rainy and foggy weather through 3D particle simulations within the Unity environments in Fig. 11

G. Limitations and Future works

Though the proposed Vid2Sim framework can support efficient training in simulation environments with fast GS-based rendering and can achieve zero-shot sim2real deployment, building each simulation environment is time-consuming as GS requires GLOMAP [42] to initialize the point cloud and the camera poses, which takes a long time to run. Therefore, we have only collected 30 environments and experiment results have shown training with more environments can lead to better performance. In the future, we will explore more efficient ways to convert the monocular videos to GS-based simulation environments and build a larger real2sim dataset with more diverse environments. We believe such large-scale environments can further benefit the training of a generalizable navigation policy and extend our pipeline to train other embodiments like humanoids and robot dogs.



Rain Simulation



Fog Simulation

Figure 11. Weather simulation with particle systems in Unity

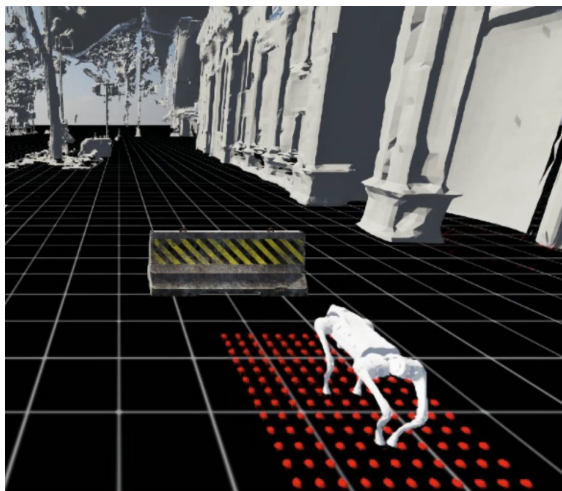
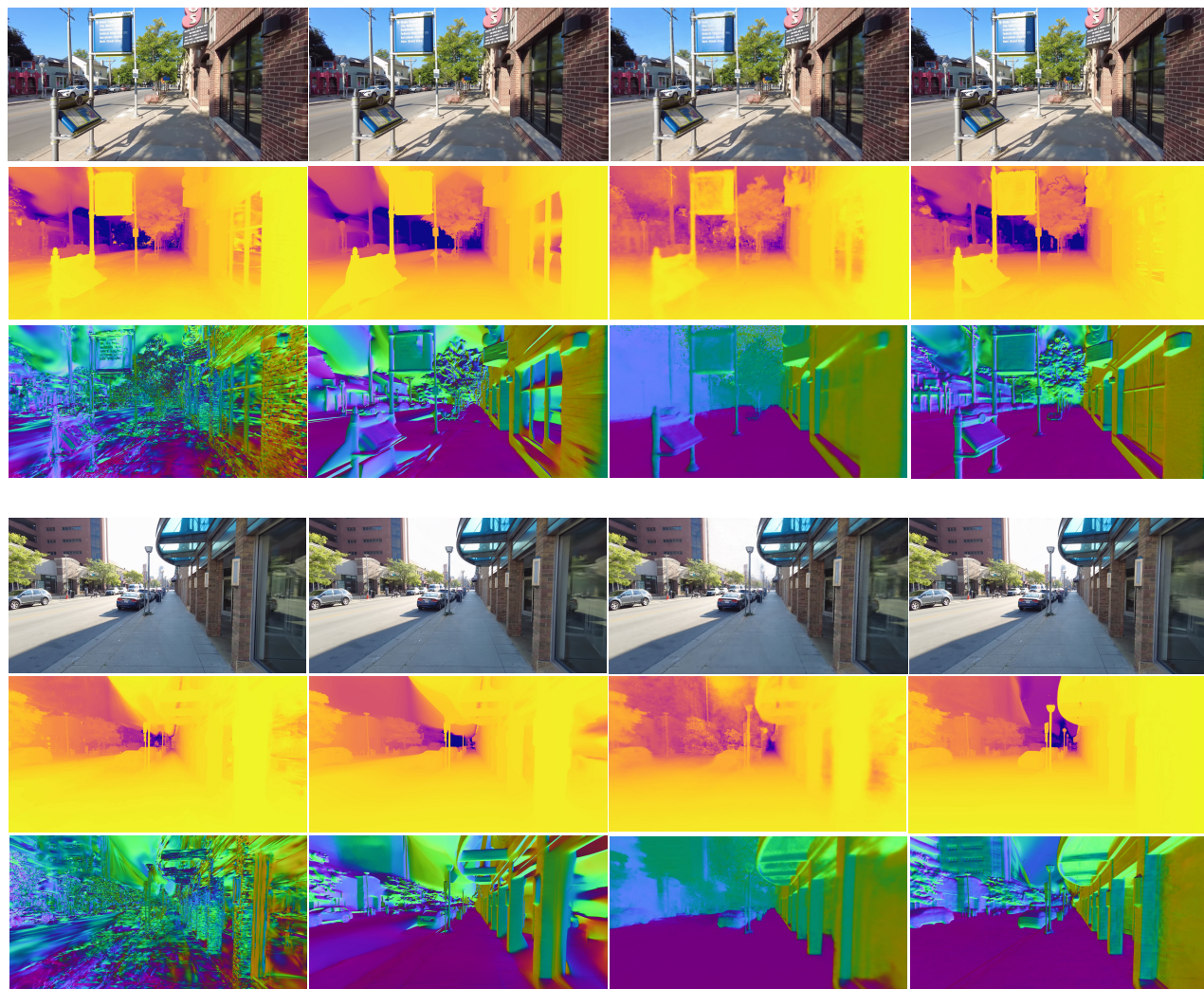


Figure 12. Extending Vid2Sim to quadruped robot Go2 in Isaac-Sim



3DGS

2DGS

Video2Game

Ours

Figure 13. Reconstruction Comparison between different methods on RGB, depth map and normal map