Habitat-GS: A High-Fidelity Navigation Simulator with Dynamic Gaussian Splatting

Ziyuan Xia[1], Jingyi Xu[1], Chong Cui[1], Yuanhong Yu[1], Jiazhao Zhang[2], Qingsong Yan[3], Tao Ni[4], Junbo Chen[4], Xiaowei Zhou[1], Hujun Bao[1], Ruizhen Hu[5], and Sida Peng[1†]

[1]Zhejiang University [2]Peking University [3]XGRIDS [4]UDeer AI [5]Shenzhen University

https://zju3dv.github.io/habitat-gs/

Abstract. Training embodied AI agents depends critically on the visual fidelity of simulation environments and the ability to model dynamic humans. Current simulators rely on mesh-based rasterization with limited visual realism, and their support for dynamic human avatars, where available, is constrained to mesh representations, hindering agent generalization to human-populated real-world scenarios. We present Habitat-GS, a navigation-centric embodied AI simulator extended from Habitat-Sim that integrates 3D Gaussian Splatting scene rendering and drivable gaussian avatars while maintaining full compatibility with the Habitat ecosystem. Our system implements a 3DGS renderer for real-time photorealistic rendering and supports scalable 3DGS asset import from diverse sources. For dynamic human modeling, we introduce a gaussian avatar module that enables each avatar to simultaneously serve as a photorealistic visual entity and an effective navigation obstacle, allowing agents to learn human-aware behaviors in realistic settings. Experiments on point-goal navigation demonstrate that agents trained on 3DGS scenes achieve stronger cross-domain generalization, with mixed-domain training being the most effective strategy. Evaluations on avatar-aware navigation further confirm that gaussian avatars enable effective human-aware navigation. Finally, performance benchmarks validate the system's scalability across varying scene complexity and avatar counts.

Keywords: Embodied AI Simulation · 3D Gaussian Splatting · Dynamic Gaussian Avatar

1 Introduction

Consider a household service robot tasked with navigating to a target location in a living room where people walk, sit, and move about freely. To navigate safely and effectively, the robot must perceive its surroundings with sufficient visual fidelity to recognize scenes and objects, and must detect and respond to dynamic human occupants for socially compliant navigation. Training such

† Corresponding author

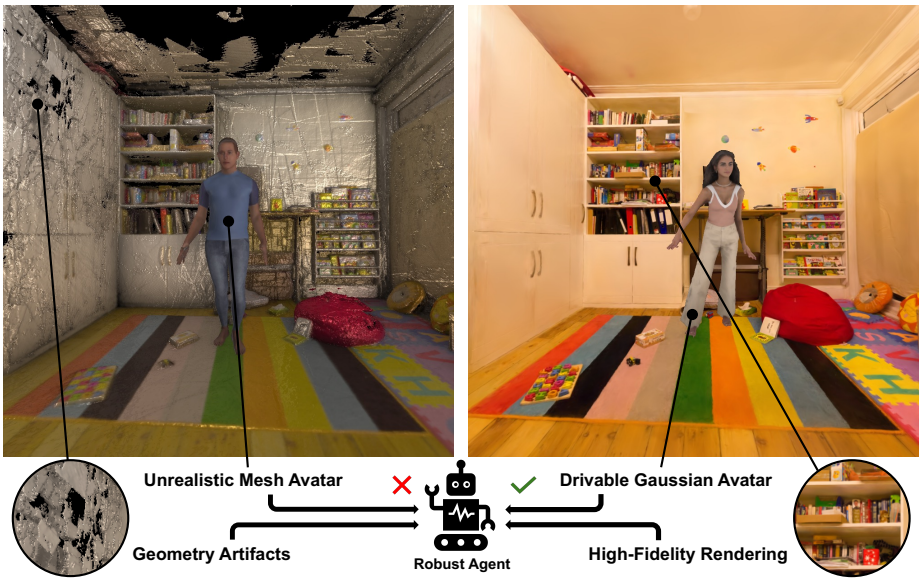


Fig. 1: Habitat-GS is a navigation-centric embodied simulation platform with 3DGS and dynamic gaussian avatars. Compared to traditional mesh-based simulators (left), our 3DGS-based simulator (right) preserves high-frequency visual details and view-dependent effects, while gaussian avatars provide realistic and dynamic human presence for human-aware navigation scenarios, thus helping train more robust agents.

embodied agents directly in the physical world is slow, expensive, potentially dangerous, and difficult to reproduce across experiments [23]. The prevailing paradigm in Embodied AI is therefore to train agents at scale in simulation and transfer the learned policies to reality [1,3,30]. This paradigm places two critical demands on the simulator: the visual realism of rendered sensor observations directly governs the effectiveness of Sim-to-Real policy transfer, and the ability to populate scenes with realistic dynamic humans determines whether agents can learn to navigate safely among people.

The leading open-source Embodied AI simulators, including Habitat-Sim [21, 23, 27], iGibson [24], AI2-THOR [9], ThreeDWorld [5], and SAPIEN [32], rely on mesh-based rasterization for visual rendering. While computationally efficient and well suited for basic geometric reasoning, the mesh representation inherently limits visual realism. Textured meshes lack high-frequency surface details, and view-dependent appearance effects such as specular reflections and subsurface scattering are difficult to reproduce faithfully via conventional rasterization. The resulting visual domain gap between simulation and reality, commonly termed the Sim-to-Real gap, has been shown to degrade the transfer of learned navigation policies to physical platforms [29]. Beyond rendering quality, native support for dynamic high-fidelity human avatars remains limited across these platforms. Habitat 3.0 [21] introduced mesh-based humanoid avatars for studying social

human-robot tasks, yet the visual quality of mesh avatars remains constrained. Most other simulators provide little or no avatar support, precluding research on navigation in human-populated environments. Finally, constructing high-quality textured mesh assets from real-world data entails labor-intensive 3D scanning, artist cleanup, and semantic annotation [26], resulting in a scalability bottleneck for expanding the diversity of training environments.

Industrial-grade platforms such as NVIDIA Isaac Sim [18], built on the Omniverse rendering stack, have recently incorporated 3D Gaussian Splatting (3DGS) [8] for enhanced visual fidelity. However, their closed-source rendering backends impede the deep customization often required for research purposes. Moreover, these platforms depend on RT Core hardware available only in RTX-series GPUs, limiting their utility on datacenter-class accelerators such as A100 and H100.

We present **Habitat-GS**, a navigation-centric embodied AI simulator upgraded from Habitat-Sim that integrates 3DGS scene rendering and drivable gaussian avatars while maintaining full compatibility with the Habitat ecosystem, including Habitat-Lab tasks, training, and evaluation APIs. Habitat-GS addresses the limitations described above along three complementary axes. *First*, we implement a 3DGS renderer that delivers real-time photorealistic rendering within the Habitat sensor pipeline, substantially narrowing the Sim-to-Real gap relative to mesh-based rendering. *Second*, we design a gaussian avatar module that represents dynamic humans as animatable gaussians. By employing pre-baked canonical gaussians with CUDA-accelerated Linear Blend Skinning, avatar deformation is driven in real time by SMPL-X [19] pose sequences without neural network inference at runtime. Combined with GAMMA-generated [34] natural motion trajectories and offline-computed proxy capsules with online NavMesh blocking, each avatar simultaneously serves as a photorealistic visual entity and an effective dynamic obstacle for navigation. *Third*, Habitat-GS supports scalable import of 3DGS assets from diverse sources. These include self-reconstructed scenes, public 3DGS datasets, and generative 3DGS pipelines like Marble [31], substantially lowering the barrier to obtaining high-quality, photorealistic scene assets.

We validate Habitat-GS through comprehensive experiments across multiple navigation paradigms. A VLM-based scene quality assessment first confirms that 3DGS scenes substantially surpass mesh scenes in rendering quality, realism, and scene diversity. On point-goal navigation, we then demonstrate that agents trained on 3DGS scenes exhibit stronger cross-domain generalization, with mixed-domain training producing the most effective strategy by combining foundational mesh-based navigation with visual robustness from GS scenes. On avatar-aware point-goal navigation, we show that training with gaussian avatars equips agents with strong collision avoidance capabilities that generalize effectively even to lower-fidelity mesh environments. Performance benchmarks further confirm the system's efficiency and scalability across varying scene scales and avatar counts.

In summary, our contributions are as follows:

- A **high-fidelity 3DGS-based embodied AI simulation platform** extended from Habitat-Sim for navigation tasks, enabling real-time photorealistic rendering and supporting scalable import from diverse 3DGS asset sources. The platform is fully open-source and maintains complete compatibility with the Habitat ecosystem.
- A **real-time drivable gaussian avatar module** that integrates photorealistic gaussian avatar rendering with collision properties, supporting natural motion and navigation-level obstacle avoidance for human-aware navigation tasks.
- **Comprehensive experimental validation** demonstrates, via VLM-based assessment and PointNav experiments, that 3DGS rendering improves agent cross-domain generalization through a mixed-domain training strategy, and that gaussian avatars enhance agent capability in human-populated scenarios, accompanied by performance benchmarks confirming system scalability.

2 Related Work

2.1 Embodied AI Simulators

Embodied AI research relies on simulation environments for large-scale parallel training and subsequent Sim-to-Real policy transfer [23,30]. We compare existing platforms along four dimensions, namely *render asset type*, *humanoid avatar support*, *platform openness*, and *hardware requirements*, in Tab. 1.

The predominant embodied AI simulators, including Habitat-Sim [21,23,27], iGibson [11,24], AI2-THOR [9], ThreeDWorld [5], and SAPIEN [32], all employ mesh-based rasterization for scene rendering. Regarding humanoid avatars, while some platforms now provide some form of mesh-based representation, such as URDF-driven articulated bodies [9,32], rigid-body tracks [11], and Unity Replicants [5], these uniformly lack the visual fidelity needed for training agents to perceive fine-grained human appearance and motion cues. Habitat 3.0 [21] offers an avatar system with deformable SMPL-X mesh avatars for human-robot social tasks, yet mesh rendering inherently constrains visual quality.

Habitat-GS inherits the Habitat ecosystem's high-performance infrastructure and training APIs while upgrading the rendering pipeline from mesh to 3DGS and natively integrating drivable gaussian avatars, uniquely combining photorealistic rendering, dynamic high-fidelity humanoids, and a mature open-source research ecosystem.

2.2 Neural Rendering

Neural Radiance Fields (NeRF) [16] demonstrated that implicit neural scene representations can synthesize photorealistic novel views from multi-view images. Subsequent works significantly improved training speed and rendering quality [2, 17]. However, NeRF's volume rendering paradigm requires per-pixel ray marching, yielding frame rates far below the real-time requirements of embodied

Table 1: Comparison of Embodied AI simulation platforms. "GPU Req." indicates hardware requirements beyond standard CUDA-capable GPUs. "No Special Req." means standard GPUs are sufficient. Habitat-GS is the only platform combining 3DGS rendering with drivable gaussian avatars while remaining fully open-source and deployable on standard datacenter accelerators.

Platform	Render Asset	Humanoid Avatar	Open-Source	GPU Req.
Habitat [21, 23, 27]	Mesh	Mesh (SMPL-X)	✓	No Special Req.
iGibson [11, 24]	Mesh	Mesh (Rigid-body)	✓	No Special Req.
AI2-THOR [9]	Mesh	Mesh (URDF)	✓	No Special Req.
ThreeDWorld [5]	Mesh	Mesh (Replicants)	✓	No Special Req.
SAPIEN [32]	Mesh	Mesh (URDF)	✓	No Special Req.
Isaac Sim [18]	Mesh+3DGS	Mesh (Sim-Ready)	Partial	Need RT Cores
Habitat-GS (Ours)	**Mesh+3DGS**	**Mesh+GS (SMPL-X)**	✓	No Special Req.

AI sensor pipelines. Moreover, its implicit representation is difficult to integrate with the explicit rendering backends employed by existing simulators.

3D Gaussian Splatting (3DGS) [8] represents scenes as collections of explicit anisotropic 3D Gaussians and renders them via differentiable tile-based rasterization, achieving real-time frame rates while maintaining rendering quality. The explicit nature of 3DGS lends itself naturally to integration with traditional graphics pipelines, for example via CUDA-OpenGL interop, and facilitates spatial editing, asset composition, and dynamic deformation. Follow-up works have extended 3DGS along multiple directions, including anti-aliasing [33] and dynamic scene modeling [15], further broadening its applicability.

Habitat-GS adopts 3DGS as its scene representation precisely because its explicit formulation and real-time rendering performance enable seamless integration into existing rasterization pipeline, while simultaneously providing high-fidelity rendering results.

2.3 Gaussian Avatars

Parametric body models such as SMPL [14] and SMPL-X [19] provide differentiable mappings from pose parameters to human body meshes, serving as the standard infrastructure for avatar animation. Building on this foundation, a growing number of work combines 3DGS with parametric body models to create high-fidelity, drivable human avatars. AnimatableGaussians [12] maps 2D Gaussians into UV space for pose-driven rendering. GaussianAvatar [7] deforms canonical-space gaussians to posed configurations via Linear Blend Skinning. GART [10] introduces gaussian articulated templates with differentiable deformation. HumanGaussian [13] leverages Score Distillation Sampling [20] for text-driven gaussian avatar generation. The shared paradigm across these methods defines gaussian attributes in a canonical space and transforms them to target poses via skinning. This approach achieves a favorable balance between visual quality and rendering speed that surpasses mesh-based avatars with limited geometric fidelity and NeRF-based avatars with insufficient rendering speed.

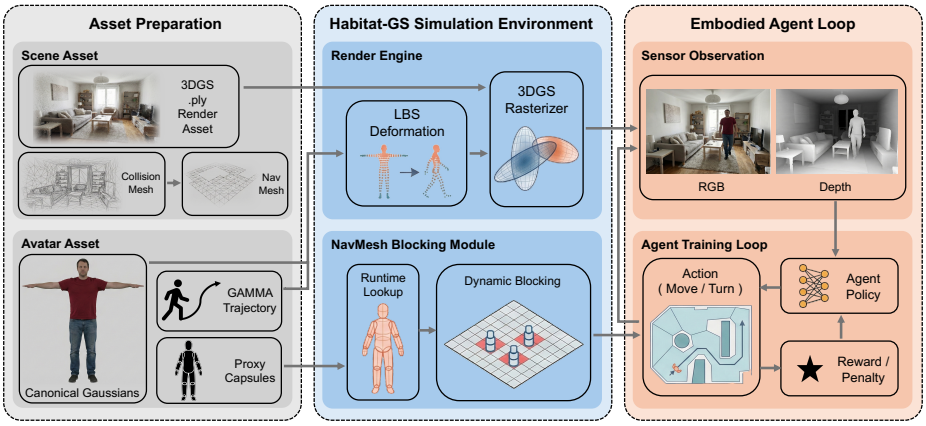


Fig. 2: System overview of Habitat-GS. From left to right: *Asset Preparation*, where 3DGS scene assets and gaussian avatar assets are prepared respectively. *Habitat-GS Simulation Environment*, where the render engine performs 3DGS rasterization for scene gaussians and LBS deformation followed by rasterization for avatar gaussians, producing RGB-D observations. The NavMesh blocking module retrieves pre-computed proxy capsules at runtime and injects them into the NavMesh for obstacle blocking. *Embodied Agent Loop*, where the agent policy consumes sensor observations, executes actions, and receives rewards shaped by both navigation success and avatar collision.

3 Habitat-GS Simulation Environment

Habitat-GS extends Habitat-Sim with two key upgrades: extending mesh-based scene rendering with 3D Gaussian Splatting (Sec. 3.2), and implementing a gaussian avatar module to drive and render avatars (Sec. 3.3). A central design principle underlying both modules is *visual–navigation decoupling*: 3DGS handles all visual rendering and traditional NavMeshes continue to govern navigation, which sidesteps the lack of explicit surface geometry inherent in the 3DGS representation. Below we discuss the overall system architecture and the detailed design of each module.

3.1 System Overview

Figure 2 illustrates the end-to-end data flow in three stages:

Asset Preparation. The scene pipeline prepares 3DGS assets in standard PLY format, together with the conventional NavMesh required by Habitat-Sim's navigation logic. The avatar pipeline produces three artifacts per identity: *first*, canonical gaussian attributes, specifically positions, SH coefficients, opacities, scales, rotations, and LBS weights, exported from a trained gaussian avatar model. *Second*, GAMMA-generated [34] motion trajectories converted to per-frame SMPL-X joint transformation matrices. *Third*, proxy capsules that approximate the avatar's per-frame collision geometry.

Habitat-GS Simulation Environment. At each frame, the system operates two parallel modules. The *render engine* dispatches scene gaussians to the CUDA rasterizer and simultaneously deforms avatar gaussians to the current pose via CUDA Linear Blend Skinning before rasterizing them with an independent renderer instance. Both color and depth outputs are transferred to OpenGL textures through CUDA–OpenGL interop. The *navigation module* retrieves the precomputed proxy capsules for current frame and injects them into the NavMesh as dynamic obstacles, enabling the path planner to produce collision-aware steps around moving avatars.

Embodied Agent Loop. The composited RGB-D observations serve as sensor inputs to the agent policy, which outputs discrete actions including move forward, turn left/right, and stop. The augmented NavMesh ensures that actions violating avatar collision boundaries are clipped, while two avatar state query APIs expose clearance distances and collision flags for reward shaping and metric computation in downstream tasks (Sec. 3.4).

3.2 3DGS Scene Rendering Integration

The core technical challenge in integrating 3DGS into Habitat-Sim lies in bridging two heterogeneous rendering backends: the Habitat sensor pipeline is built on OpenGL, whereas high-performance 3DGS rasterization relies on CUDA tile-based rendering [8]. We address this through a zero-copy CUDA–OpenGL interoperability mechanism that keeps all rendering data on the GPU throughout the entire pipeline, eliminating CPU-side data movement. At each frame, the CUDA rasterizer performs forward splatting and writes color and depth results to GPU buffers, which are then transferred directly to pre-registered OpenGL textures via intra-GPU memory copy. This enables real-time photorealistic rendering within the Habitat sensor pipeline with minimal overhead.

In practice, a Habitat-GS scene may contain both 3DGS assets and traditional mesh assets, as well as multiple gaussian avatars. To ensure correct occlusion among these heterogeneous elements, we implement a depth compositing mechanism that merges the CUDA-produced 3DGS depth with the OpenGL depth buffer via a full-screen compositing pass. For gaussian avatars, all active avatars' deformed gaussians are concatenated into a single GPU buffer and rasterized in one CUDA pass, then composited against the main framebuffer. This ensures correct depth ordering among 3DGS scenes, mesh objects, and an arbitrary number of avatars within a single frame.

3.3 Dynamic Gaussian Avatar Module

Existing gaussian avatar methods [7, 12] focus exclusively on standalone reconstruction and rendering, without coupling to simulator sensor pipelines or navigation mesh systems. To address this limitation, Habitat-GS incorporates an avatar module that supports the rendering, driving, and collision detection of gaussian avatars. We pre-bake canonical gaussian attributes offline and employ a

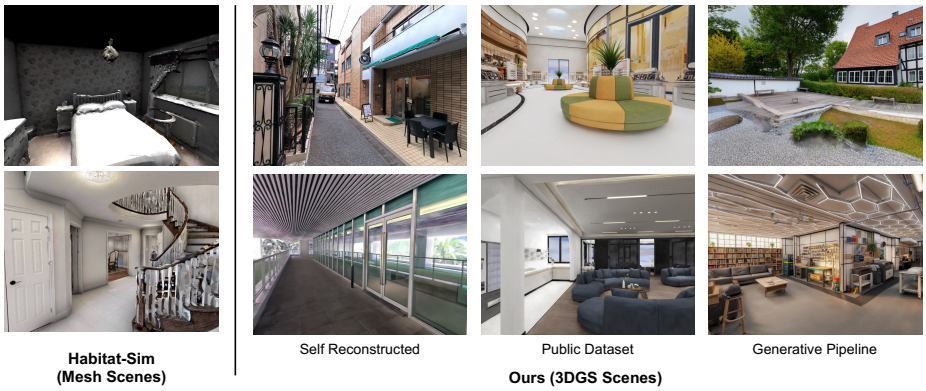


Fig. 3: Visual comparison of scene rendering. Mesh-based rendering (left) vs. our 3DGS rendering (right). Our simulator is based on 3DGS, which preserves high-frequency details and supports diverse sources of rendering assets.

lightweight CUDA LBS kernel to deform them to arbitrary SMPL-X [19] poses, thereby avoiding costly neural network inference at runtime. Below we detail how this rendering capability is combined with driving trajectory and NavMesh blocking mechanism.

GAMMA Trajectory. To drive avatars along scene-aware, physically plausible walking paths, we employ the GAMMA motion generation model [34] in an offline trajectory synthesis pipeline. Given a start point, an end point and several via points sampled on a scene's NavMesh, we first compute a shortest path connecting all waypoints. This path is then fed as guiding targets to GAMMA, which generates temporally coherent body poses coupled with root translations. For each frame, we then compute the SMPL-X joint transformation matrices via forward kinematics. At runtime, the avatar module interpolates the joint matrices and root trajectory at the current simulation time and passes the results to the CUDA LBS kernel for deformation.

Dynamic NavMesh Blocking. Since the 3DGS representation lacks a well-defined geometric surface, it is unsuitable for direct collision detection. To enable navigation-level obstacle interaction, we introduce a *proxy capsule* mechanism that decouples the avatar's visual appearance from its collision geometry. During the offline trajectory synthesis stage, we generate a set of capsule primitives from the skeletal bone segments of the SMPL-X model, and pre-compute their world-space positions for each frame of the trajectory. At runtime, capsule positions are retrieved through temporal interpolation with negligible computational cost.

A static NavMesh cannot express the time-varying occupancy of moving avatars. We therefore extend the Habitat-Sim path planner to support *dynamic capsule obstacles*. Each simulation step, the avatar manager collects the proxy

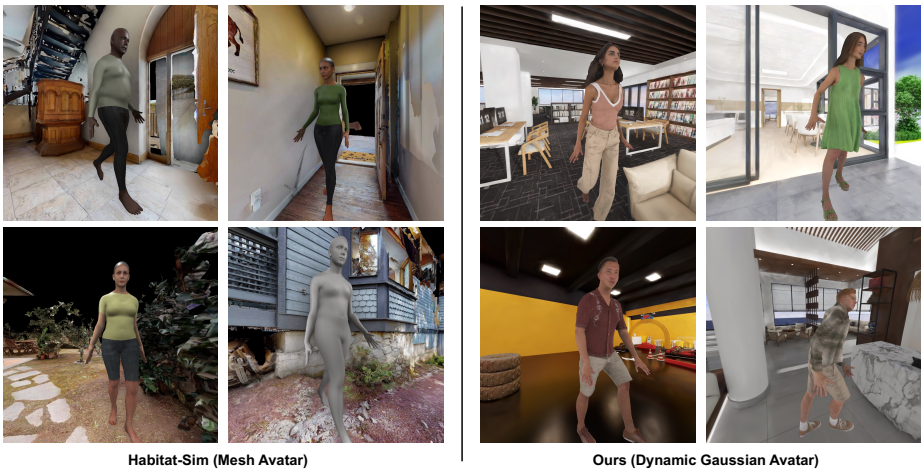


Fig. 4: Qualitative comparison of mesh avatars and gaussian avatars. Gaussian avatars exhibit significantly higher visual fidelity, preserving fine-grained details such as clothing wrinkles and hair texture.

capsules of all active avatars and injects them into the path planner as dynamic obstacles. The path planner is augmented with a step clipping routine that checks for intersection between the agent's bounding capsule and each avatar capsule, clipping agent's motion at the collision boundary if necessary. This mechanism guarantees that the agent will not pass through any avatar's body.

3.4 APIs for Habitat-Lab

Habitat-GS is designed for seamless integration with Habitat-Lab. During scene initialization, the simulator automatically loads gaussian avatar configurations from the scene description file and instantiates all avatars. At each simulation step, avatar pose updates are triggered transparently, synchronizing both visual and navigation pipeline. The sensor outputs, both RGB and depth, produced by 3DGS rendering are format-identical to those of the mesh renderer, allowing existing Habitat-Lab tasks including PointNav [1] to run on 3DGS scenes without modification.

For avatar-aware navigation tasks such as avatar-aware point-goal navigation, two additional query APIs are provided. One returns the distance to the nearest avatar capsule, and the other indicates whether a candidate step is blocked. Together, these provide the information needed for computing rewards and metrics. Researchers can leverage these APIs to design proximity-based rewards, collision penalties, and tracking metrics within the Habitat-Lab framework.

4 Experiments

We evaluate Habitat-GS along three complementary axes:

First, we quantify the quality advantage of 3DGS scenes via a VLM-based assessment and demonstrate that training on these higher-quality scenes improves agent visual robustness and cross-domain generalization on PointNav (Sec. 4.2). *Second*, we examine whether training with dynamic gaussian avatars equips agents with human-aware navigation capabilities on dynamic point-goal navigation (Sec. 4.3). *Third*, we examine whether the system remains efficient under varying scene complexity and avatar counts (Sec. 4.4). We first describe the shared experimental setup.

4.1 Experimental Setup

Datasets. For *3DGS scenes*, we combine the InteriorGS dataset [25] with additional real-world reconstructed GS scenes in a 4:1 ratio, yielding 120 scenes split into 100 for training and 20 for testing. For *mesh scenes*, we use the Habitat-Matterport 3D (HM3D) dataset [22], similarly split into 100 training and 20 test scenes. Importantly, the GS and mesh test sets are drawn from disjoint scene collections rather than two representations of the same physical spaces, so that evaluation measures cross-domain generalization.

For *gaussian avatars*, we export canonical gaussians from six trained AnimatableGaussians [12] identities, with three for training and three held out for testing. Each avatar is driven by GAMMA-generated [34] motion trajectories with pre-computed joint matrices and proxy capsules.

Evaluation Metrics. We adopt standard embodied navigation metrics. **Success Rate (SR)** measures the fraction of episodes in which the agent reaches the goal within a distance threshold. **Success weighted by Path Length (SPL)** [1] jointly captures success and path efficiency: $\mathrm{SPL} = \mathrm{SR} \times (\ell^* / \ell)$, where ℓ^* is the geodesic shortest-path length and ℓ the agent's actual path length. **Distance to Goal (DTG)** records the Euclidean distance between the agent and the goal at episode termination. Lower values indicate the agent consistently navigates closer to the target even in unsuccessful episodes.

For avatar-aware tasks, we additionally report: **Collision Rate (CR)**, the fraction of collision steps. **Personal Space Intrusion (PSI)**, which quantifies the average degree to which the agent enters the 1.0 m personal-space radius around each avatar. Higher values indicate more frequent and severe intrusions.

4.2 3DGS Scenes Train More Robust Agents

We first characterize the quality advantage of 3DGS scenes through a VLM-based assessment, and then demonstrate that this quality advantage translates to more robust navigation agents on the **PointNav** [1] task.

VLM Scene Quality Assessment. To objectively quantify the quality gap between 3DGS and mesh scene rendering, we employ Gemini 3.0 Pro [28] as an automated evaluator. We render 240 screenshots evenly from each renderer,

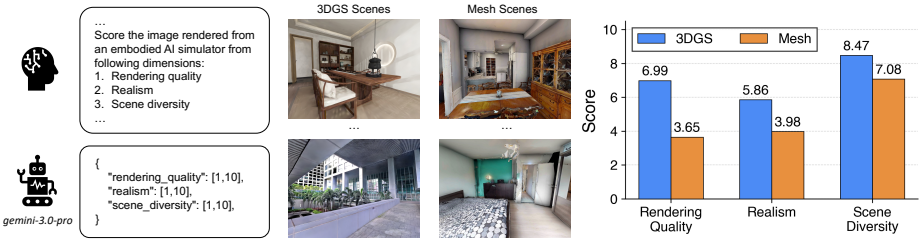


Fig. 5: VLM scene quality assessment. Gemini 3.0 Pro evaluates 240 rendered screenshots from each domain on three perceptual dimensions. GS scenes consistently outperform mesh scenes, confirming their superior visual fidelity and diversity.

divided into 48 evaluation batches, each containing 5 GS and 5 mesh images with randomized indices to blind the model of rendering source. The VLM scores each image on a 10-point scale along three dimensions: *rendering quality*, *realism*, and *scene diversity*.

As shown in Fig. 5, GS scenes substantially outperform mesh scenes across all three dimensions. These results confirm that 3DGS rendering produces higher-quality, more realistic, and more diverse training environments, establishing a strong foundation for the subsequent navigation experiments.

PointNav Setup. To validate whether the quality advantage of GS scenes translates to stronger navigation agents, we train five agent groups under different scene-domain mixtures, with training budget fixed at 5×10^7 steps: **A**: 100 mesh scenes, **B**: 100 GS scenes, **C**: 80 M + 20 G, **D**: 50 M + 50 G, and **E**: 20 M + 80 G. All agents share a unified DD-PPO [30] architecture with a ResNet [6] visual encoder and a GRU [4] policy head, receiving 256×256 RGB and depth observations, with only training scene composition varying. Each agent is evaluated on both the 20-scene mesh test set and the 20-scene GS test set.

Analysis. Table 2(a) and the training curves in Tab. 2(b,c) together reveal a clear picture of three training strategies:

Mesh-only training converges quickly but yields weak agents. As shown in Tab. 2 (b), training on 100 mesh scenes converges rapidly, yet the resulting agent plateaus at a modest success rate around 0.6 and fails to perform well on GS test set. Because mesh environments lack the visual realism of real-world scenes, the agent cannot develop robust visual representations, limiting its generalization capability. Mesh-only training is therefore not a competitive strategy.

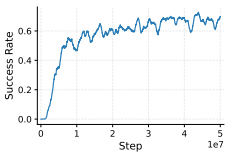
GS-only training builds stronger capability but converges too slowly. Config **B** demonstrates that 3DGS scenes can indeed enable more robust agents: the training curve in Tab. 2(c) shows that the agent reaches ∼0.8 SR on the GS training set, and it achieves higher GS test performance than the mesh-only baseline.

Table 2: PointNav results and training curves. (a) Results across different training domain mixtures. Agents are trained on varying ratios of mesh (M) and 3DGS (G) scenes and evaluated on disjoint mesh and GS test sets. SR and SPL are in %; DTG is in meters (↓). Best and Second best results per column are highlighted. (b) Training success rate curve for 100 Mesh (Config A). (c) Training success rate curve for 100 GS (Config B).

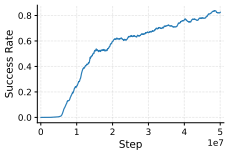
(a) Quantitative Results

	Mesh Test			GS Test		
Training Config	SR↑	SPL↑	DTG↓	SR↑	SPL↑	DTG↓
A: 100 Mesh	59.00	51.23	5.537	61.30	52.09	4.982
B: 100 GS	53.00	43.13	6.439	70.70	58.49	3.550
C: 80 M + 20 G	60.20	52.06	6.004	73.40	64.32	3.757
D: 50 M + 50 G	61.80	51.34	5.938	78.10	67.42	3.008
E: 20 M + 80 G	59.60	51.01	5.901	79.60	68.38	2.698

(b) Mesh Training



(c) GS Training



However, because GS scenes are more realistic and diverse, the agent requires considerably more steps to converge. Within the fixed budget of 5×10^7 steps, the GS-only agent has not fully converged, and its mesh test performance remains below that of Config **A**. Pure GS training is therefore not cost-effective under practical compute constraints.

Mixed-domain training is the optimal strategy. Configurations **D** (50 M + 50 G) and **E** (20 M + 80 G) demonstrate the most effective approach: a small number of mesh scenes first establish foundational navigation competence, after which training on more realistic GS scenes further strengthens the agent's ability to handle diverse and realistic environments. Config **E** achieves the best GS test performance while maintaining mesh test performance on par with the mesh-only baseline. This mixed strategy produces agents with strong cross-domain generalization, confirming that the two data modalities are complementary: mesh scenes provide efficient geometric learning, while GS scenes contribute visual robustness that transfers to more realistic deployment environments.

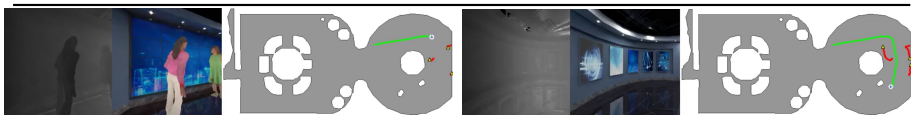
4.3 Gaussian Avatars Equip Agents with Human-Aware Navigation

We next evaluate whether gaussian avatars enable agents to navigate safely among dynamic humans. We extend the standard PointNav task by introducing three gaussian avatars walking around in scenes, and agents are supposed to reach the goal point without colliding with moving avatars.

The agents are first trained on static PointNav for 5×10^7 steps to establish basic navigation competence, and then fine-tuned for an additional 5×10^6 steps

Table 3: Avatar-aware PointNav results. Top: Two training configurations are evaluated: a GS scene baseline without avatars, and a GS scene populated with GS avatars. Agents are evaluated on both mesh and GS dynamic test sets to assess in-domain performance and cross-domain generalization. SR, SPL and CR are in %. CR is the fraction of collision steps (↓). PSI is the average personal-space intrusion (↓). **Bottom:** Visual demonstration of our avatar-aware navigation policy. Agent successfully navigates through gaussian avatars without collision or personal space intrusion.

	Mesh Scene+Avatar Test				**GS Scene+Avatar Test**			
Training Config	SR↑	SPL↑	CR↓	PSI↓	SR↑	SPL↑	CR↓	PSI↓
GS Baseline	54.80	46.98	2.521	0.075	81.80	71.35	6.713	0.092
GS Scene+GS Avatar	58.00	46.80	2.342	0.068	80.00	65.72	4.746	0.077



(a) Start of episode **(b) End of episode**

on a subset of 20 GS scenes under two configurations: **Baseline**: GS scenes, no avatars; and **GS Scene + GS Avatar**: GS scenes with gaussian avatars. Both groups are evaluated on two dynamic test sets to assess cross-domain capability: 20 mesh scenes with mesh avatars, and 20 GS scenes with gaussian avatars. We prioritize **CR** and **PSI** in avatar-aware navigation evaluation, for collision is unacceptable in real-world human-aware navigation.

Analysis. Table 3 demonstrates the clear benefit of training with gaussian avatars for human-aware navigation. *First*, compared to the avatar-free baseline, training with gaussian avatars effectively lowers the Collision Rate (CR) and Personal Space Intrusion (PSI) on the in-domain GS test set. This human-aware navigation capability is acquired with only a small number of fine-tuning steps (5×10^6, which is only 10% of the static pre-training budget), confirming that agents efficiently learn collision avoidance behaviors from the photorealistic humans and realistic motion simulations. *Second*, the agent fine-tuned with gaussian avatars also achieves lower CR (2.342% vs. 2.521%) and PSI (0.068 vs. 0.075) on the mesh test set. This improvement confirms that the high-fidelity perceptual cues learned from gaussian avatars, such as recognizing human shape, anticipating walking direction, and estimating personal space, are robust and generalize effectively even when transferred to lower-fidelity mesh environments.

4.4 Performance Benchmarks

To confirm Habitat-GS's photorealistic rendering does not compromise training speed significantly, we benchmark rendering FPS and GPU memory under varying scene scales and avatar counts. All measurements are conducted on Habitat-Lab's PointNav task on a NVIDIA RTX 4090 GPU at 256×256 resolution.

Table 4: Rendering performance and GPU memory usage. Mesh Ref. reports Habitat-Sim's native mesh rasterization on a representative HM3D scene as a baseline. (a) Varying scene Gaussian count with no avatars. (b) Varying avatar count in a medium-scale scene with approximately 2M Gaussians. Habitat-GS maintains real-time throughput under typical RL training loads with up to 5M Gaussians and 1–4 avatars.

	Mesh	**(a) Scene Scale (Gaussian Count)**						**(b) Avatar Count**				
	Ref.	300 K	500 K	1 M	3 M	5 M	7 M	0	1	2	5	10
FPS↑	163.8	159.2	148.8	120.9	82.60	51.46	44.52	94.16	75.14	57.70	37.74	24.67
Mem (GB)↓	2.930	3.299	3.373	3.567	3.723	4.091	4.425	3.917	4.197	4.457	5.513	7.497

Analysis. Table 4 demonstrates three key properties of the system. *First*, under typical RL training loads, Habitat-GS sustains real-time rendering sufficient for large-scale DD-PPO training. For medium-scale scenes with 1–2 avatars, the system still maintains >50 FPS, well above the threshold for efficient parallel training. *Second*, FPS decreases gracefully with increasing scene scale and avatar count, reflecting the favorable computational scaling of the 3DGS rasterizer [8]. Scenes within the commonly used range remain well above interactive frame rates. *Third*, GPU memory consumption grows approximately linearly with both scene gaussian count and avatar count, providing predictable resource budgeting. The throughput reduction of 3DGS rendering, relative to mesh-based rasterization, is a favorable trade-off given the improvements in visual fidelity and agent generalization demonstrated in Sec. 4.2 and Sec. 4.3.

5 Conclusion

We present Habitat-GS, a navigation-centric embodied AI simulator that upgrades the visual backbone of Habitat-Sim from mesh rasterization to 3D Gaussian Splatting, and populates environments with photorealistic drivable gaussian avatars. Habitat-GS is fully open-sourced to provide a high-fidelity, ecosystem-compatible foundation for future embodied AI research.

Limitations. A central design choice of Habitat-GS is *visual–navigation decoupling* (Sec. 3.1): 3DGS handles all visual rendering while NavMeshes are responsible for navigation logic. This intentional decoupling sidesteps the absence of explicit surface geometry in the 3DGS representation. However, it also confines physical interaction to navigation-level obstacle avoidance rather than force- or impulse-level contact. Since 3DGS is fundamentally a collection of anisotropic gaussians without inherent rigid-body properties and lacks topological connectivity, tasks requiring fine-grained physical interaction, like grasping or pushing GS-represented objects, currently fall outside the system's scope. Habitat-GS is therefore best suited for navigation tasks. Extending support to manipulation would require deeper integration with the physics engine, which we identify as a promising direction for future work.

References

1. Anderson, P., Chang, A., Chaplot, D.S., Dosovitskiy, A., Gupta, S., Koltun, V., Kosecka, J., Malik, J., Mottaghi, R., Savva, M., Zamir, A.R.: On evaluation of embodied navigation agents (2018), https://arxiv.org/abs/1807.06757
2. Barron, J.T., Mildenhall, B., Verbin, D., Srinivasan, P.P., Hedman, P.: Mip-nerf 360: Unbounded anti-aliased neural radiance fields (2022), https://arxiv.org/abs/2111.12077
3. Batra, D., Gokaslan, A., Kembhavi, A., Maksymets, O., Mottaghi, R., Savva, M., Toshev, A., Wijmans, E.: Objectnav revisited: On evaluation of embodied agents navigating to objects (2020), https://arxiv.org/abs/2006.13171
4. Cho, K., van Merrienboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., Bengio, Y.: Learning phrase representations using rnn encoder-decoder for statistical machine translation (2014), https://arxiv.org/abs/1406.1078
5. Gan, C., Schwartz, J., Alter, S., Mrowca, D., Schrimpf, M., Traer, J., Freitas, J.D., Kubilius, J., Bhandwaldar, A., Haber, N., Sano, M., Kim, K., Wang, E., Lingelbach, M., Curtis, A., Feigelis, K., Bear, D.M., Gutfreund, D., Cox, D., Torralba, A., DiCarlo, J.J., Tenenbaum, J.B., McDermott, J.H., Yamins, D.L.K.: Threedworld: A platform for interactive multi-modal physical simulation (2021), https://arxiv.org/abs/2007.04954
6. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition (2015), https://arxiv.org/abs/1512.03385
7. Hu, L., Zhang, H., Zhang, Y., Zhou, B., Liu, B., Zhang, S., Nie, L.: Gaussianavatar: Towards realistic human avatar modeling from a single video via animatable 3d gaussians (2024), https://arxiv.org/abs/2312.02134
8. Kerbl, B., Kopanas, G., Leimkühler, T., Drettakis, G.: 3d gaussian splatting for real-time radiance field rendering (2023), https://arxiv.org/abs/2308.04079
9. Kolve, E., Mottaghi, R., Han, W., VanderBilt, E., Weihs, L., Herrasti, A., Deitke, M., Ehsani, K., Gordon, D., Zhu, Y., Kembhavi, A., Gupta, A., Farhadi, A.: Ai2-thor: An interactive 3d environment for visual ai (2022), https://arxiv.org/abs/1712.05474
10. Lei, J., Wang, Y., Pavlakos, G., Liu, L., Daniilidis, K.: Gart: Gaussian articulated template models (2023), https://arxiv.org/abs/2311.16099
11. Li, C., Xia, F., Martín-Martín, R., Lingelbach, M., Srivastava, S., Shen, B., Vainio, K., Gokmen, C., Dharan, G., Jain, T., Kurenkov, A., Liu, C.K., Gweon, H., Wu, J., Fei-Fei, L., Savarese, S.: igibson 2.0: Object-centric simulation for robot learning of everyday household tasks (2021), https://arxiv.org/abs/2108.03272
12. Li, Z., Zheng, Z., Wang, L., Liu, Y.: Animatable gaussians: Learning pose-dependent gaussian maps for high-fidelity human avatar modeling. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) (2024)
13. Liu, X., Zhan, X., Tang, J., Shan, Y., Zeng, G., Lin, D., Liu, X., Liu, Z.: Humangaussian: Text-driven 3d human generation with gaussian splatting (2024), https://arxiv.org/abs/2311.17061
14. Loper, M., Mahmood, N., Romero, J., Pons-Moll, G., Black, M.J.: SMPL: A skinned multi-person linear model. ACM Trans. Graphics (Proc. SIGGRAPH Asia) **34**(6), 248:1–248:16 (Oct 2015)
15. Luiten, J., Kopanas, G., Leibe, B., Ramanan, D.: Dynamic 3d gaussians: Tracking by persistent dynamic view synthesis (2023), https://arxiv.org/abs/2308.09713

16. Mildenhall, B., Srinivasan, P.P., Tancik, M., Barron, J.T., Ramamoorthi, R., Ng, R.: Nerf: Representing scenes as neural radiance fields for view synthesis (2020), https://arxiv.org/abs/2003.08934
17. Müller, T., Evans, A., Schied, C., Keller, A.: Instant neural graphics primitives with a multiresolution hash encoding. ACM Transactions on Graphics **41**(4), 1–15 (Jul 2022). https://doi.org/10.1145/3528223.3530127, http://dx.doi.org/10.1145/3528223.3530127
18. NVIDIA: Isaac Sim, https://github.com/isaac-sim/IsaacSim
19. Pavlakos, G., Choutas, V., Ghorbani, N., Bolkart, T., Osman, A.A.A., Tzionas, D., Black, M.J.: Expressive body capture: 3d hands, face, and body from a single image (2019), https://arxiv.org/abs/1904.05866
20. Poole, B., Jain, A., Barron, J.T., Mildenhall, B.: Dreamfusion: Text-to-3d using 2d diffusion (2022), https://arxiv.org/abs/2209.14988
21. Puig, X., Undersander, E., Szot, A., Cote, M.D., Yang, T.Y., Partsey, R., Desai, R., Clegg, A.W., Hlavac, M., Min, S.Y., Vondruš, V., Gervet, T., Berges, V.P., Turner, J.M., Maksymets, O., Kira, Z., Kalakrishnan, M., Malik, J., Chaplot, D.S., Jain, U., Batra, D., Rai, A., Mottaghi, R.: Habitat 3.0: A co-habitat for humans, avatars and robots (2023), https://arxiv.org/abs/2310.13724
22. Ramakrishnan, S.K., Gokaslan, A., Wijmans, E., Maksymets, O., Clegg, A., Turner, J., Undersander, E., Galuba, W., Westbury, A., Chang, A.X., Savva, M., Zhao, Y., Batra, D.: Habitat-matterport 3d dataset (hm3d): 1000 large-scale 3d environments for embodied ai (2021), https://arxiv.org/abs/2109.08238
23. Savva, M., Kadian, A., Maksymets, O., Zhao, Y., Wijmans, E., Jain, B., Straub, J., Liu, J., Koltun, V., Malik, J., Parikh, D., Batra, D.: Habitat: A platform for embodied ai research (2019), https://arxiv.org/abs/1904.01201
24. Shen, B., Xia, F., Li, C., Martín-Martín, R., Fan, L., Wang, G., Pérez-D'Arpino, C., Buch, S., Srivastava, S., Tchapmi, L.P., Tchapmi, M.E., Vainio, K., Wong, J., Fei-Fei, L., Savarese, S.: igibson 1.0: a simulation environment for interactive tasks in large realistic scenes (2021), https://arxiv.org/abs/2012.02924
25. SpatialVerse Research Team, M.T.I.: Interiorgs: A 3d gaussian splatting dataset of semantically labeled indoor scenes. https://huggingface.co/datasets/spatialverse/InteriorGS (2025)
26. Straub, J., Whelan, T., Ma, L., Chen, Y., Wijmans, E., Green, S., Engel, J.J., Mur-Artal, R., Ren, C., Verma, S., Clarkson, A., Yan, M., Budge, B., Yan, Y., Pan, X., Yon, J., Zou, Y., Leon, K., Carter, N., Briales, J., Gillingham, T., Mueggler, E., Pesqueira, L., Savva, M., Batra, D., Strasdat, H.M., Nardi, R.D., Goesele, M., Lovegrove, S., Newcombe, R.: The replica dataset: A digital replica of indoor spaces (2019), https://arxiv.org/abs/1906.05797
27. Szot, A., Clegg, A., Undersander, E., Wijmans, E., Zhao, Y., Turner, J., Maestre, N., Mukadam, M., Chaplot, D., Maksymets, O., Gokaslan, A., Vondrus, V., Dharur, S., Meier, F., Galuba, W., Chang, A., Kira, Z., Koltun, V., Malik, J., Savva, M., Batra, D.: Habitat 2.0: Training home assistants to rearrange their habitat (2022), https://arxiv.org/abs/2106.14405
28. Team, G., Anil, R., Borgeaud, S., et al.: Gemini: A family of highly capable multimodal models (2025), https://arxiv.org/abs/2312.11805
29. Tobin, J., Fong, R., Ray, A., Schneider, J., Zaremba, W., Abbeel, P.: Domain randomization for transferring deep neural networks from simulation to the real world (2017), https://arxiv.org/abs/1703.06907
30. Wijmans, E., Kadian, A., Morcos, A., Lee, S., Essa, I., Parikh, D., Savva, M., Batra, D.: Dd-ppo: Learning near-perfect pointgoal navigators from 2.5 billion frames (2020), https://arxiv.org/abs/1911.00357

31. World Labs: Marble: A multimodal world model (11 2025), https://www.worldlabs.ai/blog/marble-world-model
32. Xiang, F., Qin, Y., Mo, K., Xia, Y., Zhu, H., Liu, F., Liu, M., Jiang, H., Yuan, Y., Wang, H., Yi, L., Chang, A.X., Guibas, L.J., Su, H.: Sapien: A simulated part-based interactive environment (2020), https://arxiv.org/abs/2003.08515
33. Yu, Z., Chen, A., Huang, B., Sattler, T., Geiger, A.: Mip-splatting: Alias-free 3d gaussian splatting (2023), https://arxiv.org/abs/2311.16493
34. Zhang, Y., Tang, S.: The wanderings of odysseus in 3d scenes (2022), https://arxiv.org/abs/2112.09251

Appendix

This appendix provides supplementary information, extended evaluations, and implementation specifics to further support the main text. Specifically, Sec. A offers a more detailed illustration of the Habitat-GS architecture, elaborating on system components and data flow. Sec. B presents additional qualitative results, including extended visualizations of our 3DGS scenes, dynamic avatars and navigation episodes. Sec. C covers further experimental details, comprehensive task definitions, and extended navigation benchmark results.

A Habitat-GS Architecture Details

This section provides a detailed description of the Habitat-GS architecture, which serves as a further refinement of Sec. 3 in the main text.

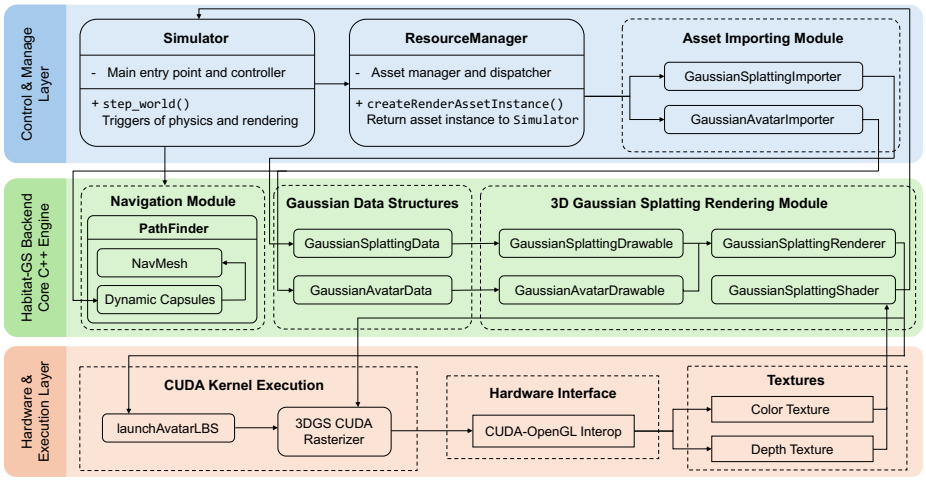


Fig. 6: System architecture of Habitat-GS. The system adopts a "visual–navigation decoupling" design principle, separating the visual rendering modules handled by the CUDA-based 3DGS rasterizer and LBS deformation, from the navigation module managed by the traditional NavMesh and injected proxy capsules. This allows for photo-realistic agent observations without modifying the core Habitat-Sim navigation logic.

Habitat-GS is built as a non-intrusive extension to Habitat-Sim [21,23,27]. As illustrated in Fig. 6, the system data flow is systematically organized into interconnected module stages within the simulation environment. First, during asset initialization, scene and avatar data are parsed by custom `Importer` modules and stored in CPU `Data` containers before being attached to the Habitat SceneGraph via extended `Drawable` node interfaces. At each simulation frame, the visual rendering module performs CUDA tile-based rasterization for 3DGS [8]

and CUDA Linear Blend Skinning for dynamic gaussian avatars. The computed color and depth outputs are routed to the OpenGL sensor pipeline via a zero-copy CUDA–OpenGL interoperability mechanism. Synchronized with visual rendering, the dynamic navigation module retrieves pre-computed proxy capsules corresponding to the current motion timestamp and injects them into the `PathFinder` NavMesh as temporary spatial obstacles. Such data flow ensures that high-fidelity rendering outputs seamlessly feed into agent sensor observations, while collision constraints continuously inform the underlying path planning logic, thereby maintaining robust performance and full compatibility within the existing Habitat ecosystem.

A.1 Asset Management

Habitat-Sim's native asset pipeline relies on Magnum and primarily supports mesh formats (e.g., `.glb`, `.obj`). To accommodate 3DGS assets, we implement a custom `GaussianSplattingImporter` in C++ to parse `.ply` binary formats. It extracts essential gaussian attributes including canonical positions μ, spherical harmonics (SH) coefficients $\mathbf{c}$, opacities α, scales $\mathbf{s}$, and rotations $\mathbf{q}$. This customized importer unifies assets from diverse sources, such as self-reconstructed scenes, public 3DGS datasets, and generative 3DGS pipelines [31], into a standardized data container `GaussianSplattingData`, thus integrating with Habitat's existing asset management module.

For gaussian avatars, we provide scripts to export the results of existing avatar methods like AnimatableGaussians [12] and GaussianAvatar [7] into the canonical representation in `.npz` format. `GaussianAvatarImporter` then reads this unified format, which contains canonical gaussian attributes alongside Linear Blend Skinning weights $\mathbf{W}$ and inverse bind matrices $\mathbf{B}^{-1}$, and subsequently stores them in a `GaussianAvatarData` container.

A.2 Drawable Extension and CUDA–OpenGL Interoperability

Habitat's rendering pipeline is organized around a drawable abstraction in which each renderable entity in the scene graph implements a draw callback. To integrate 3DGS into the Habitat-Sim SceneGraph, we introduce new `Drawable` abstractions: `GaussianSplattingDrawable` for static scenes and `GaussianAvatarDrawable` for dynamic avatars. These classes encapsulate the 3DGS rendering logic and coexist with traditional mesh objects via the same rendering interface.

The core rendering challenge lies in bridging the CUDA tile-based rasterizer with the OpenGL-based Habitat sensor pipeline. While a naive solution would be continuously copying rendered frames between GPU and CPU, this introduces severe latency that is unacceptable for real-time simulation. We tackle this through zero-copy CUDA–OpenGL interoperability in a *Map–Render–Unmap* mechanism. At initialization, OpenGL color and depth textures are registered as CUDA-accessible resources. During each frame's rendering pass, these textures are dynamically mapped into the CUDA address space. The CUDA rasterizer executes forward splatting and writes the RGB and depth outputs directly into

the mapped buffers. After unmapping, the textures are immediately available to the OpenGL sensor pipeline without any data transfer between the CPU and GPU, ensuring real-time performance. Finally, a full-screen depth compositing shader merges the 3DGS depth buffer with the OpenGL depth buffer, correctly resolving occlusions among 3DGS scenes, mesh objects, and multiple gaussian avatars.

A.3 Avatar Kinematics and CUDA LBS

Offline Export. To avoid the substantial latency of neural network inference at runtime, the gaussian avatar module relies on a pre-baked canonical representation combined with a highly optimized CUDA Linear Blend Skinning kernel. During offline preparation, canonical gaussian attributes (positions μ, rotations $\mathbf{q}$, scales $\mathbf{s}$, properties $\mathbf{c}$ and α, alongside LBS weights $\mathbf{W}$), as well as pose-dependent joint transformation matrices $\mathbf{T}_j$ generated by GAMMA [34], are exported and serialized into driving `.pkl` files.

Runtime Deformation. At runtime, the simulator retrieves and interpolates these pre-computed joint transformation matrices $\mathbf{T}_j$ based on the target timestep. A parallelized CUDA kernel then performs the explicit LBS deformation. For each gaussian point i, its canonical position μ_i is transformed to the posed space position μ_i' by blending the transformations of the nearest joints according to the pre-computed skinning weights $w_{i,j}$:

$$\mu_i' = \sum_j w_{i,j} \mathbf{T}_j \mu_i \tag{1}$$

Similarly, the canonical rotation quaternion $\mathbf{q}_i$ is adjusted using dual-quaternion blending parameterized by the same weights. This explicit deformation paradigm avoids costly neural network operations, maintaining high frame rates even when rendering multiple dynamic avatars simultaneously.

A.4 Dynamic Navigation Module

Because the 3DGS representation lacks explicit geometric surfaces, direct collision detection is computationally infeasible. To enable navigation-level physics like obstacle avoidance, we introduce a *proxy capsule mechanism* (see Sec. 3.3 of the main text).

Offline pre-computation. During the trajectory synthesis stage, a temporary URDF representation is generated from the rest-pose joint positions of the SMPL/SMPL-X [14, 19] model, containing one capsule per skeletal bone segment with radius proportional to bone length. For each frame of the generated trajectory, forward kinematics along the URDF joint chain produces world-space capsule endpoints $(\mathbf{p}_0, \mathbf{p}_1, r)$ for every bone. The resulting capsule sequences are stored as a $[T, C, 7]$ tensor with T being the number of frames and C the number of capsules per skeleton, in the driving `.pkl` file.

Runtime Injection. At runtime, the avatar module loads the pre-computed capsule data at initialization and retrieves current capsule positions via temporal linear interpolation. This reduces the per-frame cost to a simple array index and interpolation with no forward kinematics evaluation. All capsules from all active avatars are merged into a single $[N, 7]$ array containing $\mathbf{p}_0$, $\mathbf{p}_1$, and radius per capsule for downstream navigation mesh.

This design achieves an effective balance. The Gaussian representation delivers photorealistic rendering, while the lightweight pre-computed capsule proxy provides efficient collision primitives for navigation planning without imposing heavy runtime computational overhead.

B Simulation Visualization

B.1 3DGS Scene and Avatar Visualization

In this section, we provide additional qualitative results of the Habitat-GS simulation environment. Fig. 7 presents a diverse set of 3DGS scenes populated with high-fidelity dynamic gaussian avatars, further demonstrating the photorealistic rendering capabilities and the support for diverse assets of our system.

B.2 Navigation Episodes

We present qualitative visualizations of Habitat-Lab's navigation episodes conducted within the Habitat-GS environment. Figure 8 shows example episodes on five different navigation tasks from our experiments, illustrating how agents trained on Habitat-GS navigate through 3DGS environments while interacting with dynamic gaussian avatars. These visualizations prove Habitat-GS's compatibility with Habitat-Lab and further demonstrate that the agents learn meaningful navigation behaviors after training on our simulator.

C Experimental Details and Further Results

C.1 Additional Results on Static Navigation

In the main text (Sec. 4.2), the five PointNav training configurations are compared under a fixed budget of $5{\times}10^7$ steps. As noted, this budget is sufficient for mesh-only training (Config **A**) to converge, but GS-heavy configurations, particularly Config **B** (100 GS), have not yet fully converged due to the greater visual diversity of 3DGS scenes. To provide a fair comparison that disentangles training efficiency from final capability, we extend all five configurations to $1{\times}10^8$ steps so that every configuration reaches full convergence.

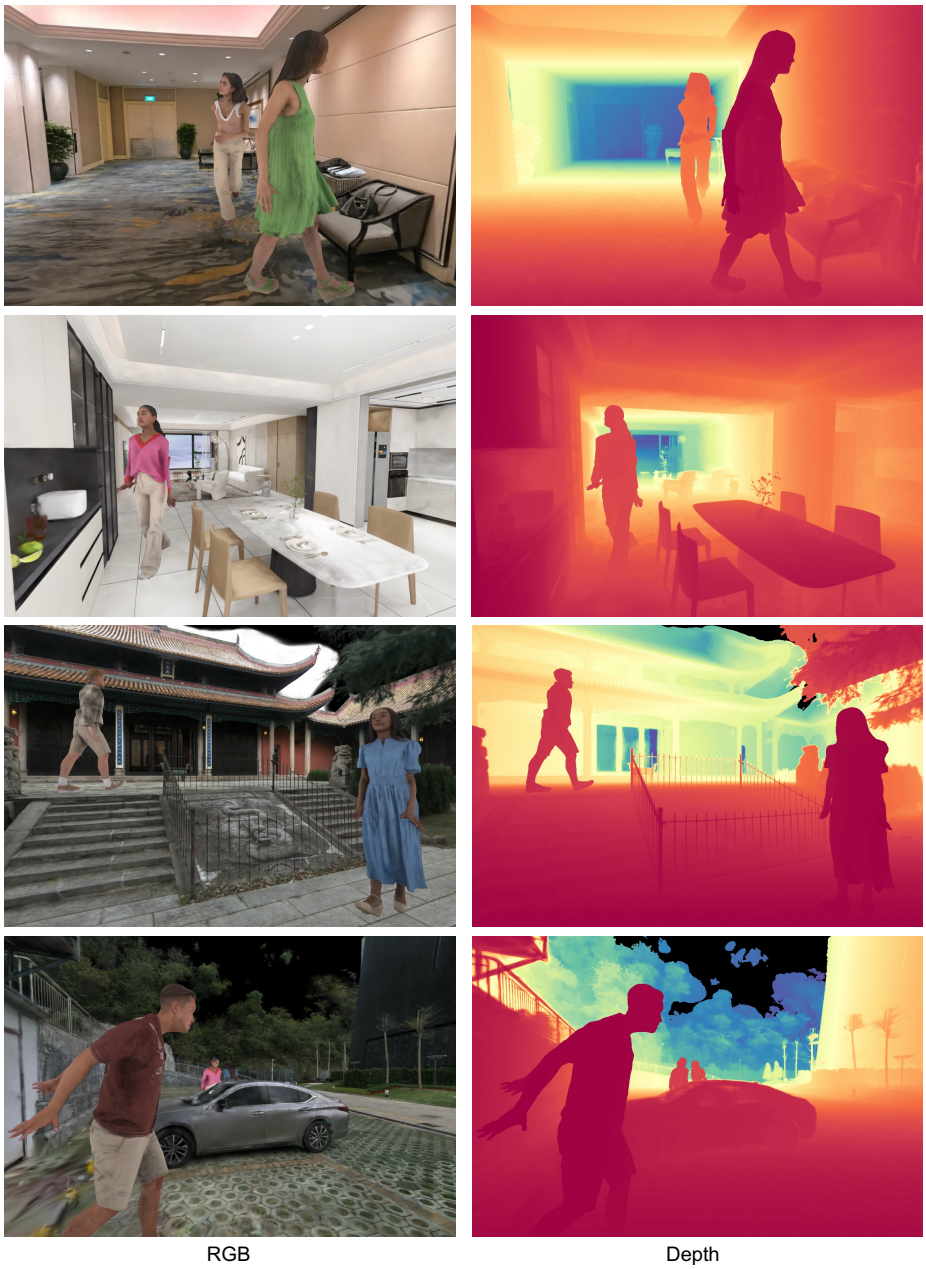


Fig. 7: Additional visualizations of 3DGS scenes and gaussian avatars. Habitat-GS supports real-time rendering of diverse, large-scale indoor and outdoor environments with photorealistic quality, while simultaneously integrating high-fidelity drivable human avatars to facilitate human-aware embodied AI research.

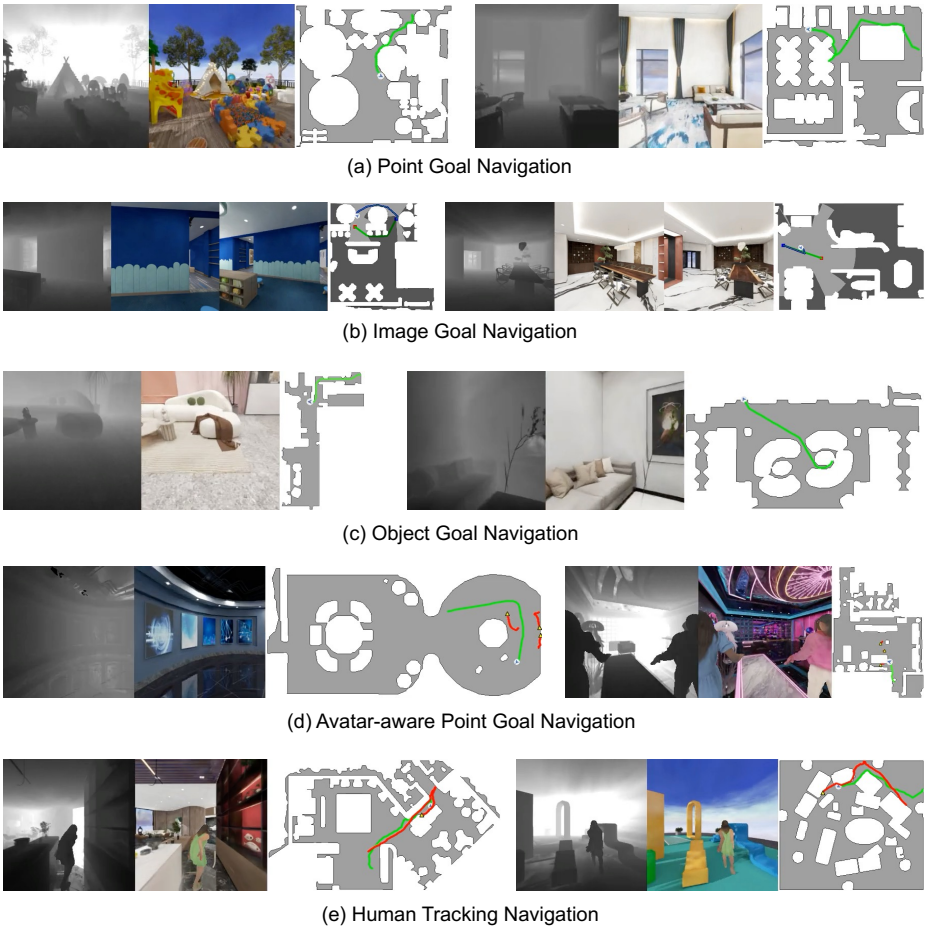


Fig. 8: Qualitative visualization of navigation episodes. Example episodes from our navigation experiments conducted in the Habitat-GS environment. The visualizations illustrate agent trajectories in 3DGS scenes, demonstrating learned navigation behaviors including goal-directed path planning and human-aware obstacle avoidance.

Table 5 presents the resulting Success Rate, SPL, and Distance to Goal on both the mesh and GS test sets. We also provide an **Average** column that reports the mean of each metric across the mesh and GS test sets, which measures the overall cross-domain generalization capability and comprehensive performance of trained agents.

Analysis. Table 5 together with Tab. 2 in the main text reveal the following conclusions:

Table 5: Fully converged PointNav results (1×10^8 steps). All five training configurations are trained to full convergence. SR and SPL are in %; DTG is in meters (↓). Best and Second best results per column are highlighted. **Average** column measures the cross-domain generalization performance of trained agents.

	Mesh Test			GS Test			Average		
Training Config	SR↑	SPL↑	DTG↓	SR↑	SPL↑	DTG↓	SR↑	SPL↑	DTG↓
A: 100 Mesh	62.80	54.75	5.211	61.70	52.06	5.187	62.25	53.41	5.199
B: 100 GS	59.80	49.27	5.911	77.60	66.96	2.777	68.70	58.12	4.344
C: 80 M + 20 G	63.00	53.52	5.658	77.60	67.60	3.351	70.30	60.56	4.505
D: 50 M + 50 G	62.80	53.26	5.748	79.50	69.43	2.911	71.15	61.35	4.330
E: 20 M + 80 G	62.10	52.39	5.612	80.20	69.37	2.906	71.15	60.88	4.259

Mesh-only training excels on mesh but fails on realistic scenes. Config **A** achieves the highest mesh-test SPL and the lowest mesh-test DTG, confirming that mesh scenes provide efficient geometric learning for mesh environments. However, this advantage is strictly in-domain. When evaluated on the more diverse and realistic GS test set, Config **A** falls dramatically behind all other configurations, with the lowest GS SR (61.70) and GS SPL (52.06). This large gap demonstrates that mesh-only training produces agents whose visual representations are narrowly tuned to the synthetic appearance of mesh scenes and do not generalize to realistic visual conditions.

GS-only training is the better single-source strategy. Config **B** achieves substantially higher GS-test performance than Config **A** (SR 77.60 vs. 61.70, SPL 66.96 vs. 52.06), while its mesh-test performance is only slightly lower (SR 59.80 vs. 62.80). This indicates that the visual robustness acquired from realistic GS scenes transfers reasonably well to mesh environments, whereas the converse is not true. Therefore, if training scenes are restricted to a single source, GS scenes are the superior choice for training generalizable navigation agents.

Mixed-domain training remains optimal. Even after full convergence, Configs **D** (50 M+50 G) and **E** (20 M+80 G) dominate the Average columns. They tie for the best average SR and Config **D** achieves the best average SPL, while Config **E** attains the best average DTG. These configurations maintain competitive mesh-test performance close to the mesh-only baseline, while simultaneously reaching the highest GS-test scores. This further confirms the complementary role of the two data modalities observed in the main text: mesh scenes provide foundational navigation competence through efficient geometric learning, and GS scenes further enhance agent robustness and generalization through their greater visual diversity and realism. Mixed-domain training is therefore the most effective strategy for producing agents with strong cross-domain generalization.

C.2 Avatar-aware PointNav Task Definition

Our avatar-aware PointNav task (Sec. 4.3) keeps the standard Habitat PointNav [1] goal and action interface, but augments the reward with avatar-aware safety signals. Concretely, the policy still solves a point-goal navigation problem with the standard discrete action space {`stop`, `move_forward`, `turn_left`, `turn_right`}, and the observation space remains the standard PointNav observation bundle used in our experiments, namely RGB-D egocentric observations together with the point-goal signal in the agent frame. The key difference is that dynamic avatars are injected into the simulator as both rendering entities and proxy capsules, so they affect both the agent's sensor observations and the reward returned to DD-PPO [30].

Task interpretation. The task is best defined as standard PointNav under dynamic social constraints. The goal specification and success condition remain those of PointNav, but avatars become moving obstacles in the transition model and contribute a dense safety penalty in the reward. This makes the problem substantially different from static-scene PointNav: a successful policy must not only reach the goal efficiently, but do so while respecting the personal space of nearby avatars.

Signed clearance to avatars. Let c_t denote the minimum signed clearance between the robot capsule and all active avatar proxy capsules at time t. By construction, $c_t > 0$ means the agent is separated from the nearest avatar, $c_t = 0$ means touching, and $c_t < 0$ means overlap. If no avatar proxy is active, the simulator returns $+\infty$, in which case the avatar-related penalty is zero. A binary collision flag is recorded using a small numerical tolerance $\varepsilon_{\text{col}} = 10^{-5}$, i.e.,

$$\text{collision}_t = \mathbb{I}[c_t \leq \varepsilon_{\text{col}}]. \tag{2}$$

Personal intrusion function. The task tracks a per-step *personal intrusion* value that measures how deeply the agent enters the avatar's personal space. The total intrusion can be written as a single three-piece function:

$$I_t = \begin{cases} 0, & c_t \geq d_{\text{int}}, \\ d_{\text{int}} - c_t, & 0 \leq c_t < d_{\text{int}}, \\ d_{\text{int}}, & c_t < 0. \end{cases} \tag{3}$$

In the PointNav configuration in our experiments, the personal-space threshold is set to $d_{\text{int}} = 1.0$ m and the critical threshold is set to $d_{\text{crit}} = 0.5$ m. Outside the personal-space radius, the agent incurs no intrusion. Once the clearance falls below d_{int}, the intrusion grows linearly as the agent moves closer to the avatar. When overlap occurs, the intrusion saturates at d_{int} rather than increasing without bound, which keeps the metric numerically stable and easy to interpret across episodes. Thus, I_t serves as a descriptive measure of social-space violation.

Two-stage avatar penalty. The avatar-related reward does not use the linear intrusion value directly. Instead, we design a smoother two-stage penalty as a function of signed clearance:

$$P(c_t) = \begin{cases} 0, & c_t \geq d_{\text{int}}, \\ p_1 \left(\frac{d_{\text{int}} - c_t}{d_{\text{int}} - d_{\text{crit}}} \right)^{\alpha_1}, & d_{\text{crit}} \leq c_t < d_{\text{int}}, \\ p_1 + (p_{\text{col}} - p_1) \left(1 - \left(\frac{\max(c_t, 0)}{d_{\text{crit}}} \right)^{\alpha_2} \right), & c_t < d_{\text{crit}}. \end{cases} \tag{4}$$

In our experiments, the parameters are $p_1 = 0.1$, $\alpha_1 = 2.0$, $\alpha_2 = 4.0$, $p_{\text{col}} = 0.6$, and the final penalty is capped by $p_{\max} = 0.6$. The reward measure returned by the simulator is therefore

$$r_t^{\text{avatar}} = -\min(P(c_t), p_{\max}). \tag{5}$$

This design is deliberate. The first segment provides a weak, smooth repulsion near the boundary of personal space, which discourages the agent from grazing avatars while preserving stable optimization. The second segment becomes much steeper inside the critical zone, strongly penalizing unsafe proximity and saturating at a fixed collision penalty when contact or overlap occurs. The two branches are continuous at $c_t = d_{\text{crit}}$, and the second branch also saturates smoothly at $c_t \leq 0$.

Total reward. The base navigation term is still the standard PointNav progress reward:

$$r_t^{\text{prog}} = d_{t-1}^{\text{goal}} - d_t^{\text{goal}}, \tag{6}$$

where d_t^{goal} is the geodesic distance to the navigation goal at step t. The avatar-aware PointNav reward measure linearly combines progress and avatar safety:

$$r_t^{\text{measure}} = \lambda_{\text{prog}} r_t^{\text{prog}} + \lambda_{\text{avatar}} r_t^{\text{avatar}}, \tag{7}$$

with weights $\lambda_{\text{prog}} = 1.0$ and $\lambda_{\text{avatar}} = 1.0$ in our experiments. Finally, the RL environment wrapper adds the standard Habitat task-level slack and success terms:

$$r_t = r_t^{\text{measure}} + r_{\text{slack}} + \mathbb{I}[\text{success}_t] R_{\text{succ}}. \tag{8}$$

Therefore, the agent is rewarded for shortening the geodesic distance to the goal, simultaneously discouraged from entering an avatar's personal space, and strongly penalized when it gets dangerously close or collides.

C.3 Human Tracking Experiment

Beyond avatar-aware PointNav, we further evaluate Habitat-GS on a self-defined *human tracking* task (TrackNav), where the agent has no fixed goal position but must continuously follow a moving avatar at an appropriate distance and orientation. Unlike PointNav, the goal is dynamic: the agent must stay within a

prescribed *distance band* behind the target avatar while maintaining visual contact. This tests whether the simulator can support more complex, long-horizon human-aware navigation tasks.

Tracking definition. A tracking step is considered successful when three conditions are satisfied simultaneously:

1. **Distance band.** The Euclidean distance between the agent and the target avatar's center lies within $[d_{\min}, d_{\max}]$, where $d_{\min} = 1.2$ m and $d_{\max} = 2.5$ m in our experiment.
2. **View cone.** The angle between the agent's forward heading and the direction toward the target is at most $\theta_{\text{view}} = 45^\circ$, ensuring the target remains within the agent's frontal field of view.
3. **Rear sector.** The angle between the avatar's backward-facing direction and the vector from the avatar to the agent is at most $\theta_{\text{rear}} = 60^\circ$, constraining the agent to follow from behind rather than from the side or front.

We denote the binary indicator as $\text{track}_t = \mathbb{I}[\text{all three conditions hold at step } t]$. The cumulative *track rate* is $\text{TR} = \sum_t \text{track}_t \,/\, T$, where T is the total number of steps.

Reward design. The TrackNav reward is composed of several dense shaping terms.
(i) Approach-to-band progress. Let e_t denote the unsigned distance from the current agent-to-target distance to the nearest boundary of the tracking band $[d_{\min}, d_{\max}]$, i.e., $e_t = 0$ when the agent is inside the band. The approach reward is

$$r_t^{\text{approach}} = \lambda_{\text{app}}(e_{t-1} - e_t), \tag{9}$$

with $\lambda_{\text{app}} = 0.8$, rewarding the agent for reducing the gap to the band.
(ii) In-band reward. When the agent is inside the distance band, a Gaussian-shaped bonus encourages it to stay near the band center:

$$r_t^{\text{band}} = r_{\text{peak}} \left[\eta + (1-\eta) \, \frac{\exp\!\left(-\frac{1}{2}(\delta/\sigma)^2\right) - w_{\text{edge}}}{1 - w_{\text{edge}}} \right], \tag{10}$$

where δ is the distance from the band center, $\sigma = 0.85 \times \frac{1}{2}(d_{\max} - d_{\min})$, $\eta = 0.58$ is the edge ratio, $w_{\text{edge}} = \exp(-\frac{1}{2}(h/\sigma)^2)$ with h the half-span, and $r_{\text{peak}} = 0.36$. The reward is zero outside the band.
(iii) Avatar collision penalty. The same two-stage penalty used in avatar-aware PointNav (Sec. C.2) is applied, driven by the minimum clearance from all proxy capsules. Due to the closer desired operating range of tracking, the penalty magnitudes are scaled down: $p_1 = 0.03$, $p_{\text{col}} = 0.12$, $\alpha_1 = 2.0$, and $\alpha_2 = 3.0$.
(iv) Orientation penalties. When the agent is inside the distance band but fails to satisfy the view or rear constraint, per-step penalties of $p_{\text{view}} = 0.10$ and

Table 6: Human tracking (TrackNav) cross-validation results. Agents are trained under two configurations and cross-evaluated on both test sets. TR is the cumulative track rate (%). CC is the average agent-avatar collision count (↓).

	Mesh Test		**GS Test**		**Average**	
Training Config	TR↑	CC↓	TR↑	CC↓	TR↑	CC↓
Mesh Scene+Mesh Avatar	38.40	12.14	17.78	11.72	28.09	11.93
GS Scene+GS Avatar	28.42	6.280	19.84	5.080	24.13	5.680

$p_{\text{rear}} = 0.10$ are applied respectively, discouraging the agent from drifting to the avatar's side or front while maintaining distance.
(v) Anti-circling shaping. A dense radial/tangential decomposition of each step rewards motion toward the target (weight 0.9) and penalizes tangential orbiting (weight 0.45), preventing the agent from circling the avatar without closing distance.
(vi) Track bonuses. A per-step bonus $b_{\text{step}} = 0.08$ is given for each successful tracking step. An additional streak bonus grows linearly with the number of consecutive tracking steps, up to a cap of 50 steps, with maximum bonus $b_{\text{streak}} = 0.12$:

$$r_t^{\text{streak}} = b_{\text{streak}} \cdot \frac{\min\big(\text{streak}_t, 50\big)}{50}. \tag{11}$$

Experimental setup. All agents use the same DD-PPO [30] architecture as in the PointNav experiments, receiving 256×256 RGB-D observations and the 4-dimensional tracking sensor. We train two agents separately: **(1) Mesh Scene + Mesh Avatar**: trained on mesh scenes with mesh humanoid avatars, and **(2) GS Scene + GS Avatar**: trained on GS scenes with gaussian avatars. Each agent is then cross-evaluated on mesh test scenes with mesh avatars and GS test scenes with gaussian avatars to assess in-domain performance and cross-domain generalization.

Analysis. Table 6 presents the cross-validation results for the TrackNav task. During evaluation, we prioritize **CC** over **TR**, as colliding with humans may pose severe safety risks in real-world scenarios.

Mesh-trained agents achieve higher tracking rates at the cost of unacceptable collision levels. While the agent trained on mesh scenes and mesh avatars attains a higher average TR, this advantage comes at the expense of drastically elevated collision counts. Its average CC (11.93) is more than double that of the GS-trained agent (5.68). This indicates that the mesh-trained agent has learned a distorted navigation strategy that aggressively pursues the target while largely

ignoring avatar proximity, trading safety for tracking performance. Such behavior is unacceptable in real-world deployment. A household service robot that repeatedly collides with people during navigation would pose severe safety hazards and undermine user trust. Therefore, despite its slightly higher TR, mesh-trained agent cannot be considered a feasible policy for human-tracking navigation.

GS-trained agents learn safer and more generalizable human-aware policies. The agent trained on GS scenes and gaussian avatars exhibits significantly lower collision counts across both test sets, demonstrating that it has learned to better coexist with humans and navigate with a more human-friendly strategy. This substantially improved safety comes with only a moderate reduction in tracking capability, and notably, the GS-trained agent achieves a higher TR on the GS test set than its mesh-trained counterpart, indicating strong in-domain tracking performance. Furthermore, the GS-trained agent generalizes well across domains. Its performance remains stable when evaluated on the mesh test set, whereas the mesh-trained agent suffers a dramatic TR drop from 38.40 on mesh test to 17.78 on GS test, revealing poor cross-domain transferability. The GS-trained policy therefore offers a better balance of safety and effectiveness, making it the more practical and deployable choice for real-world human-aware navigation.

C.4 Text Prompt for VLM Evaluation

We provide the complete text prompt used for the Gemini 3.0 Pro [28] scene quality assessment described in Sec. 4.2 of the main text. The prompt is designed to objectively elicit 1-10 scores on rendering quality, realism, and scene diversity from exactly 10 images randomly sampled from mesh and GS rendering results.

```
You are an expert visual evaluator for embodied AI simulation imag
es.

You will receive exactly 10 rendered scene images from embodied AI
simulators.

For EACH image, score 3 criteria on 1-10 scale (higher is better):
1) rendering_quality:
   Visual fidelity (sharpness, blur, artifacts, texture quality, g
   eometry consistency).
2) realism:
   How realistic and natural the rendered scene looks compared wit
   h real-world scenes.
3) scene_diversity:
   How distinct this image is compared with the other 9 images in
   this same batch (layout variety, scene types, objects, visual
   appearance variety).

Important constraints:
```

```
- Do NOT infer or mention rendering methods, engines, or dataset
names.
- Do NOT output markdown. Return strict JSON only.

Required JSON format:
{
  "images": [
    {
      "image_position": 1,
      "rendering_quality": <integer in [1,10]>,
      "realism": <integer in [1,10]>,
      "scene_diversity": <integer in [1,10]>
    },
    ...
    {
      "image_position": 10,
      "rendering_quality": <integer in [1,10]>,
      "realism": <integer in [1,10]>,
      "scene_diversity": <integer in [1,10]>
    }
  ],
  "brief_reason": "<max 60 words>"
}
```