Buildee: A 3D Simulation Framework for Scene Exploration and Reconstruction with Understanding

Clémentin Boittiaux Vincent Lepetit
LIGM, École Nationale des Ponts et Chaussées, IP Paris, CNRS, France

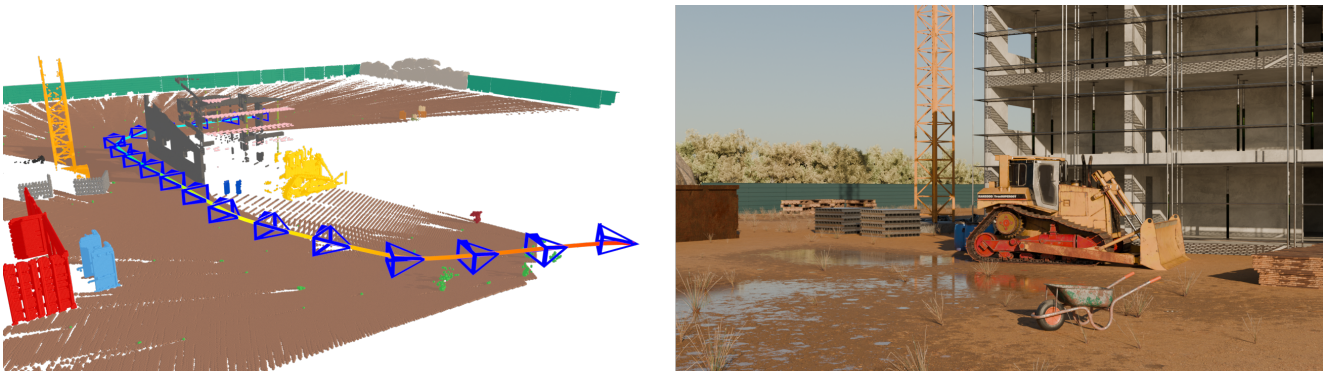


Figure 1. **Exploring the scene with Buildee.** On the left, a camera is exploring a construction site, collecting RGB, depth and semantic observations. Using our dynamic environment, researchers can benchmark various computer vision tasks against ground truth, including next-best-view and semantic point cloud segmentation. On the right, we show an example of a high-quality RGB observation obtained with Buildee using Blender's rendering engine. Beyond construction sites, Buildee allows the development and the evaluation of exploration and reconstruction algorithms in realistic and complex environments.

Abstract

We introduce Buildee, a 3D simulation framework designed to benchmark scene exploration, 3D reconstruction, and semantic segmentation tasks in both static and dynamic environments. Built as a Python module on top of Blender, Buildee leverages its advanced rendering capabilities to generate realistic RGB, depth, and semantic data while enabling 2D / 3D point tracking and occlusion checking. Additionally, we provide a procedural generator for construction site environments and baseline methods for key computer vision tasks. Through Buildee, we establish a standardized platform for evaluating scene understanding algorithms in realistic settings. Our code is publicly available at github.com/clementinboittiaux/buildee.

1. Introduction

Supervising a construction site requires a comprehensive understanding of its current state, including tracking material deliveries, locating machinery, and monitoring the progress of buildings. A powerful way to enhance site management is through a digital twin—a continuously updated 3D virtual representation of the construction site, accurately placing machines, materials, and structures over time.

To build such a digital twin, one possible approach is to deploy an autonomous Unmanned Aerial Vehicle (UAV) that periodically scans the entire site, capturing images to reconstruct an up-to-date 3D model of the scene. This process involves solving several key robotics and computer vision challenges: next-best-view estimation to efficiently explore the environment while minimizing redundancy [7, 8, 11], 3D reconstruction to recover the site's geometry [22, 24], and point cloud segmentation to semantically classify objects within the scene [13, 16].

These tasks are fundamental across a wide range of domains, including autonomous navigation [3, 6] and environmental monitoring [20]. However, developing and benchmarking algorithms for these challenges in real-world scenarios can be costly, time-consuming, and logistically complex. A controlled, repeatable, and scalable simulation environment is essential to accelerate research and ensure fair comparisons between different approaches.

To address this need, we introduce Buildee, a 3D simulation framework implemented as a Python module based on Blender. Buildee enables embodied agents to navigate com-

Simulator	Dynamic scenes	Articulated objects	Realistic shaders	Interactive	2D point tracking	3D point tracking
Gazebo [1]	✓	✓	✗	✓	✗	✗
CARLA [6]	✓	~	✓	✓	✓	✗
Gibson Env [25]	✗	✓	✗	✓	✗	✗
BlenderProc [5]	✓	✓	✓	✗	✓	✗
Habitat [15, 21, 23]	~	✓	✗	✓	✗	✗
Genesis [2]	✓	✓	✓	✓	✗	✗
Buildee (ours)	✓	✓	✓	✓	✓	✓

Table 1. **Comparing Buildee to other simulation frameworks for computer vision.** By enabling to track 3D points on dynamic objects, and by leveraging the capabilities of Blender rendering engines, our simulation framework enables to benchmark scene exploration and reconstruction tasks such as next-best-view and semantic point cloud segmentation in realistic environments.

plex, dynamic 3D environments, capture observations, and compute key performance metrics. This facilitates direct comparisons between various approaches, including next-best-view estimation, 3D reconstruction, semantic segmentation of point clouds, and integrated all-in-one methods.

Our framework offers several essential features for benchmarking these tasks. For instance, it allows to develop and evaluate scene exploration and reconstruction methods. It provides depth maps and semantic segmentation to the user while detecting collisions between the virtual camera and the environment. Another of the key features of Buildee is the generation of high-quality RGB images with advanced visual effects by leveraging Blender's rendering engines. Moreover, Buildee can sample 3D points in the scene and provide their trajectories in both full 3D and occlusion-aware 2D view projections for dynamic and static objects alike. These capabilities, highlighted in Fig. 1, enable precise monitoring of scene coverage for next-best-view strategies by determining which regions have been observed and which remain unseen, and provide ground-truth segmented point clouds for evaluating 3D reconstruction and semantic segmentation algorithms.

Many state-of-the-art methods for 3D exploration rely on PyTorch3D [19] for rendering and tracking observed points [7, 8], but PyTorch3D only provides low-quality rendering, as it simply projects textures without accounting for reflections or other effects. In contrast, our Blender-based solution offers high-quality rendering with realistic lighting and shading. Additionally, our framework seamlessly integrates dynamic and static environments created directly in Blender, simplifying both environment generation and manipulation. Users can thus easily design custom scenarios, and semantic information parsing is fully automated.

In addition to the simulation framework, we provide a procedural generator that creates diverse construction site environments. The generated scenes are created using different objects, some of which were downloaded from Sketchfab and are under CC license. We created the other object models ourselves, which we also put under CC license.

To demonstrate the utility of our simulator, we implement baseline methods for aforementioned computer vision tasks. Specifically, we provide the codebase for simple random walk strategies as a reference for exploration and and 3D reconstruction with semantic segmentation, leveraging Buildee's ground truth depth and segmentation maps for the latter. These baselines can be easily adapted by the users to implement their own methods. We evaluate these baselines using key performance metrics on different seeds of our procedural construction site generator, reporting the area under the curve (AUC) of surface coverage for next-best-view, Chamfer distance for 3D reconstruction, and mean intersection over union (mIoU) for semantic segmentation. These results serve as initial benchmarks for future methods tackling the same challenges.

2. Related work

Recent advances in simulation frameworks [2, 15] and procedural scene generation [9, 17, 18] have made it easier to create realistic and interactive environments. Simulation enables large-scale data collection with precise ground-truth annotations, while procedural generation enables the creation of diverse environments, reducing overfitting on fixed datasets and improving generalization in deep learning models. In this section, we first compare Buildee with existing simulation frameworks and then briefly outline its targeted computer vision and robotics tasks.

2.1. Simulation frameworks

Various simulation frameworks have been developed to support robotics and computer vision research, each with different focuses and limitations. Table 1 provides a comparative overview of existing simulators [1, 5, 6, 15, 21, 23, 25].

One key limitation of existing simulators is the lack of 3D point tracking, which is essential for benchmarking

next-best-view methods. Without this capability, it is difficult to determine which parts of a scene have been observed during an exploration task. Most next-best-view approaches rely on PyTorch3D for rendering and point tracking [7, 8, 11], but PyTorch3D offers only basic texture rendering without advanced effects such as reflections or realistic lighting. In contrast, our Blender-based solution benefits from high-quality rendering and physics-based illumination, enabling more realistic simulations.

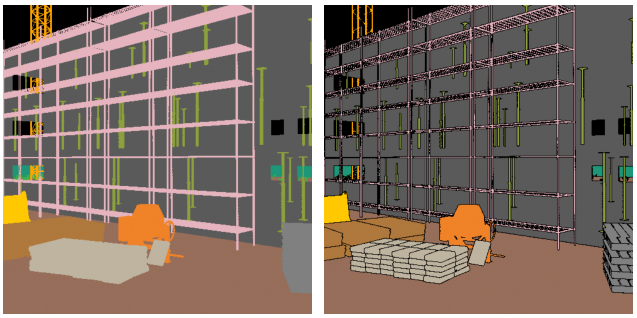
Some simulators, such as BlenderProc [5], share similarities with our approach, particularly in leveraging Blender for rendering. However, BlenderProc is designed primarily for dataset generation rather than interactive simulation, making it unsuitable for tasks requiring agent interaction. Other frameworks, such as Genesis, focus on high-speed physics simulation and generative data synthesis but do not emphasize next-best-view exploration or semantic segmentation. CARLA [6] is widely used in autonomous driving research [4], offering high-quality urban environments but lacking the necessary tools for next-best-view evaluation. Similarly, Gibson [25] and Habitat [15, 21] support embodied AI research with realistic indoor environments but do not provide explicit 3D point tracking or high-quality modular rendering engines.

Another limitations of all of these simulators is that, with the exception of BlenderProc, none of them have an accessible interface for designing and manipulating 3D scenes, making it harder to design custom environments.

2.2. Target tasks

Buildee is designed to support several key computer vision and robotics tasks that are essential for scene understanding and autonomous navigation. In this section, we provide a comprehensive overview of these core tasks, highlighting their significance, challenges.

Next-best-view. This task addresses the challenge of estimating the optimal camera viewpoint to maximize coverage gain during scene exploration. More specifically, given a set of previously acquired observations, next-best-view algorithms aim to estimate the next 6-DoF camera pose that maximizes scene coverage while minimizing redundancy. This task is fundamental for autonomous exploration applications [7, 8, 11]. Evaluating next-best-view algorithms requires the ability to determine scene coverage, as well as reliable collision detection capabilities that Buildee specifically provides through its 3D point tracking and occlusion checking mechanisms. By maintaining a complete history of observed points throughout an exploration sequence, Buildee enables quantitative evaluation of next-best-view methods using metrics such as AUC of surface coverage.



(a) Low sensitivity (b) High sensitivity

Figure 2. **Semantic segmentation sensitivity.** The parameter `segmentation_sensitivity` controls the precision of object boundaries in the semantic map. Low sensitivity (left) provides precise boundaries but may introduce noise at object edges, while high sensitivity (right) creates cleaner boundaries but may miss thin structures.

3D reconstruction. 3D reconstruction involves recovering the geometric structure of a scene from sensor observations, usually in the form of RGB or RGB-D images. This field has evolved from classical Structure-from-Motion (SfM) [22] and Multi-View Stereo methods to recent neural approaches such as Neural Radiance Fields [14] and 3D Gaussian Splatting [10]. In this work, consistent with the output of most SfM pipelines, we consider 3D reconstruction estimates in the form of point clouds. In this case, evaluating 3D reconstruction is done by comparing the estimated point cloud to the ground truth point cloud provided by Buildee. A common evaluation metric for this task is the Chamfer distance [12], which measures the average distance between the points in the predicted and ground truth point clouds.

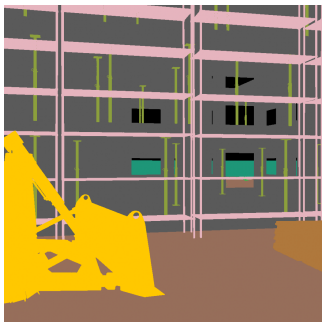
Point cloud semantic segmentation. Point cloud semantic segmentation assigns a semantic label to each point in a 3D point cloud [3]. Unlike instance segmentation or panoptic segmentation, which also identify individual object instances, semantic segmentation focuses solely on categorizing points according to their object label. Evaluating semantic segmentation algorithms requires semantic labels for each point. Buildee addresses this need by providing point clouds with semantic labels directly sampled from the surface of objects in the scene. The most commonly used evaluation metric for this task is mIoU, which measures the overlap between predicted and ground truth segmentation masks for each label.

3. Framework

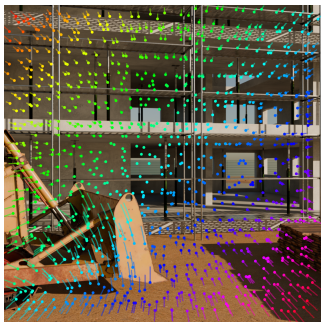
In this section, we first introduce the key features of Buildee from a user's perspective. We then explore the technical implementation details that enable these capabilities.

(a) RGB image

(b) Depth map



(c) Semantic segmentation map



(d) 2D / 3D point tracking

Figure 3. **Observations modalities.** Buildee provides RGB images (Fig. 3a), metric depth maps (Fig. 3b) and semantic segmentation maps (Fig. 3c). Additionally, it provides a dynamic semantic point cloud of the scene with occlusion checking from the camera's viewpoint, enabling 2D / 3D point tracking (Fig. 3d). The rendering settings can be adjusted in the Blender file for either better visual quality or faster rendering. The density of the point cloud can be adjusted during scene initialization.

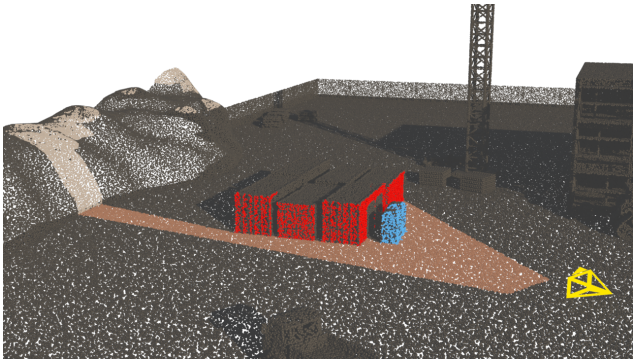


Figure 4. **3D point cloud with occlusion checking.** Buildee performs visibility checks for each 3D point. This is achieved by casting rays from the camera to the scene using dynamic and static BVH trees. Visible points are shown with their semantic colors.

3.1. Key features

Buildee is designed as a user-friendly Python module with a strong focus on simple and intuitive user experience, while providing powerful capabilities for scene exploration and understanding. It offers a comprehensive set of tools for researchers working on next-best-view algorithms, 3D reconstruction, and semantic segmentation.

Scene loading and initialization. Creating a simulation instance and loading a Blender scene requires just a few lines of code:

```
from buildee import Simulator
sim = Simulator(
    blend_file="scene.blend",
    points_density=1.0,
    segmentation_sensitivity=0.1
)
```

The constructor accepts several key parameters:

- `blend_file` : path to the Blender file containing the 3D scene.
- `points_density` : controls the density of sampled 3D points on object surfaces—these points are used for 2D / 3D point tracking and visibility analysis. Higher values create denser point clouds.
- `segmentation_sensitivity` : controls the precision of object boundaries in semantic segmentation. Values range from 0 to 1, as shown in Fig. 2.

During initialization, Buildee performs several important operations: sampling 3D points on all object surfaces, establishing semantic labels based on object names, configuring rendering pipelines for RGB, depth, and segmentation, and building acceleration structures for occlusion testing.

Scene rendering. Rendering RGB images, depth maps, and semantic segmentation maps is achieved through a single function call:

```
rgb, depth, seg = sim.render()
```

The `render()` function returns three image arrays depicted in Figs. 3a to 3c:

- `rgb` : the RGB image rendered using Blender's rendering engine—Eevee or Cycles, as configured in the Blender file.
- `depth` : the depth map in metric units, where each pixel's value is the distance from the camera.
- `seg` : the semantic segmentation map where each pixel contains a label ID.

The semantic information can be accessed through:

```
label_id = seg[0, 0]  # ID at pixel (0, 0)
label_name = sim.labels[label_id]
```

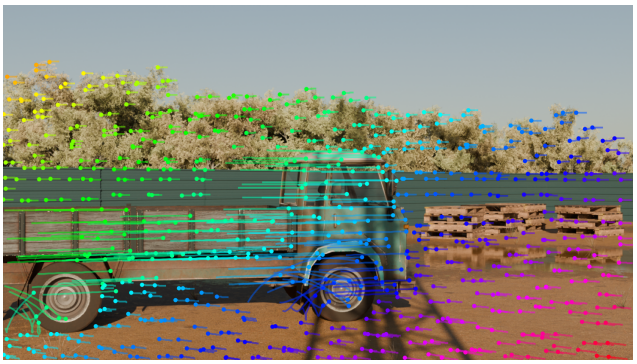


Figure 5. **Tracking 2D / 3D points.** Both the truck and the camera are moving right. Buildee is able to track 3D points of both static objects in the background and dynamic objects such as the truck. Since the truck moves faster than the camera, its displacement in the image is in the opposite direction of the background, while its rotating wheels create cycloid patterns.

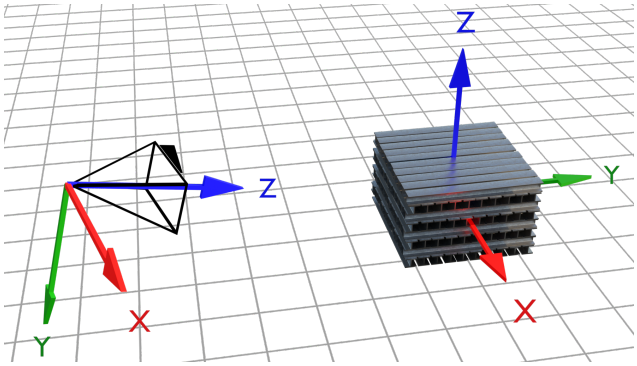


Figure 6. **Buildee's coordinate system.** Buildee shares the same coordinate system as COLMAP [22]. The world coordinate system is Z-axis up. The camera coordinate system is Z-axis forward and Y-axis down—compared to Blender's camera system, there is a 180° rotation along the X-axis.

Dynamic point cloud ground truth. One of Buildee's most powerful feature is its ability to provide a complete representation of the 3D scene with semantic labels and visibility information:

```
pcl, labels, mask = sim.compute_point_cloud()
```

This function returns three arrays:

- `pcl` : the positions of all 3D points in world coordinates.
- `labels` : semantic label IDs for each point.
- `mask` : boolean mask indicating which points are visible from the current camera view, illustrated in Fig. 4.

The function also accepts an optional `update_mask` parameter which, when set to True, updates an internal observation history, allowing the framework to track which points have been observed throughout an exploration sequence. This is particularly valuable for next-best-view algorithms, as it enables measurement of scene coverage. For dynamic objects, point positions are automatically updated when objects move, enabling consistent 3D point tracking through time. When used in conjunction with occlusion check, and by projecting points onto the image plane, this also allows for 2D point tracking as shown in Figs. 3d and 5.

Camera navigation. Buildee provides flexible camera navigation functions with built-in collision detection:

```
# Set camera pose with a 4x4 transform matrix
collided = sim.set_world_from_camera(matrix4x4)

# Convenience functions for common movements
sim.move_camera_forward(1.0)
sim.move_camera_right(0.5)
sim.move_camera_down(-0.3)
sim.turn_camera_right(15.0, degrees=True)
```

The navigation functions return a boolean indicating whether a collision occurred. If a collision is detected, the camera remains at its previous position. Additionally, given a spawn volume defined in the Blender file, our framework enables random camera respawn in this volume:

```
sim.respawn_camera()
```

Buildee also provides access to the camera's intrinsic parameters. Similary to standard computer vision tools like COLMAP [22], our framework uses the coordinate system conventions shown in Fig. 6.

Dynamic scenes. Buildee supports fully dynamic scenes with animated objects:

```
sim.step_frame()  # Step one frame forward
```

This advances the simulation time, updating all animated objects in the scene. The framework automatically tracks both static and dynamic objects separately, ensuring that dynamic objects are properly handled for point tracking, occlusion testing, and semantic segmentation.

Unprojecting points. Buildee provides a utility function to unproject the acquired depth map back into 3D world coordinates:

```
points3D = sim.depth_to_world_points(depth)
```

This function can be used to simulate the acquisition of a dense 3D point cloud such as those obtained using RGB-D cameras.

3.2. Under the hood

To provide these features efficiently, Buildee relies on several key technical Blender components as we describe here.

Semantic segmentation. At initialization, we extract labels from the base names of each object in the scene, *e.g.*, `Building` and `Building.001` both have the `Building` label. Then, in compositing, we link each object to a Cryptomatte node and assign the label ID to the matte's color. Because the matte outputs an antialiased image, we apply a threshold to it—this is the `segmentation_sensitivity` parameter explained in Fig. 2. All individual mattes are then merged into a single semantic segmentation map based on the depth map—the closest matte is selected for each pixel.

Point cloud generation. To provide a 3D point cloud ground truth representation of the scene, we use Geometry Nodes to sample points across each object's surface. During generation, we make a distinction between static and dynamic objects—points of static objects are stored in world coordinates, while points of dynamic objects are stored in object coordinates because they will need to be updated on each frame. Thus, dynamic points are transformed in world coordinates in the `compute_point_cloud` function, depending on the object's current transformation.

Dynamic scenes occlusion checking. In order to perform occlusion check for dynamic objects efficiently, Buildee maintains two separate Bounding Volume Hierarchy (BVH) trees: one for static objects and one for dynamic objects. The static BVH tree is built once during initialization, while the dynamic BVH tree is rebuilt as needed for each call to `compute_point_cloud` . Dynamic objects are detected by checking if the object has either keyframes or Blender animation drivers. To identify visible points, we first project all 3D points onto the image plane using the camera's intrinsic and extrinsic parameters, and then perform ray-casting from the camera to each point in the frustum—this reduces the number of ray-casting operations.

Random camera spawn. A practical feature for training next-best-view algorithms is the ability to spawn the camera at a random location. During initialization, Buildee looks for a `CameraSpawn` object in the scene. If found, we use Geometry Nodes to convert this object into a volume, and then sample random points in this volume to obtain a set of camera spawn points. The `CameraSpawn` object is a simple mesh with any shape designed to enclose the camera spawn volume. If this object is not found, the only spawn point is the initial camera position.

Rendering pipeline. The framework configures a node-based rendering pipeline in Blender to generate RGB, depth, and semantic data in a single render pass. RGB images are rendered using Blender's rendering engines, while depth information is extracted through the Z-pass and stored in the alpha channel of the image. This RGB-D image, as well as the semantic segmentation map derived from the Cryptomatte nodes are both saved using the EXR format without any compression. EXR files enable high precision depth values, and they are also faster to save and load than PNG or JPEG files.

4. Tools and examples

In this section, we first present a procedural construction site generator that can be used as a dataset to train and evaluate embodied agents. We also showcase the versatility of Buildee through concrete examples that demonstrate its capabilities for 3D exploration and mapping and 3D semantic scene reconstruction.

4.1. Procedural construction site environment

Along with our simulation framework, we provide a procedural construction site generator that can be used to train and evaluate embodied agents. The generator is implemented in Blender using the Geometry Nodes feature. It allows to procedurally generate scenes with different presets, including different object densities, random patterns and scene sizes. The scene is created using different objects, some of which were downloaded from Sketchfab and are under CC license, and we created the other object models ourselves. The seed number can be set manually to ensure reproducibility when evaluating different computer vision tasks on the scene. Examples of generated scenes are shown in Fig. 7.

4.2. Exploration

In order to validate the use of our framework for the exploration and mapping problem, we considered two simple exploration baselines based on a random walk strategy. Users can refer to these easy-to-use baselines as examples and modify the code to suit their needs.

Experimental setup. For our first random walk strategy, our agent performs 500 exploration steps, roaming through the scene and updating Buildee's internal 3D point observation history. For each step, the agent chooses one of 5 possible actions: move forward, move left, move right, turn left and turn right. For the second strategy, which we call "modified random walk", the agent moves straight until it encounters an obstacle, then rotates in a random direction free of obstacles, and iterates. For both strategies, we consider 3D points to be visible only if they were observed from

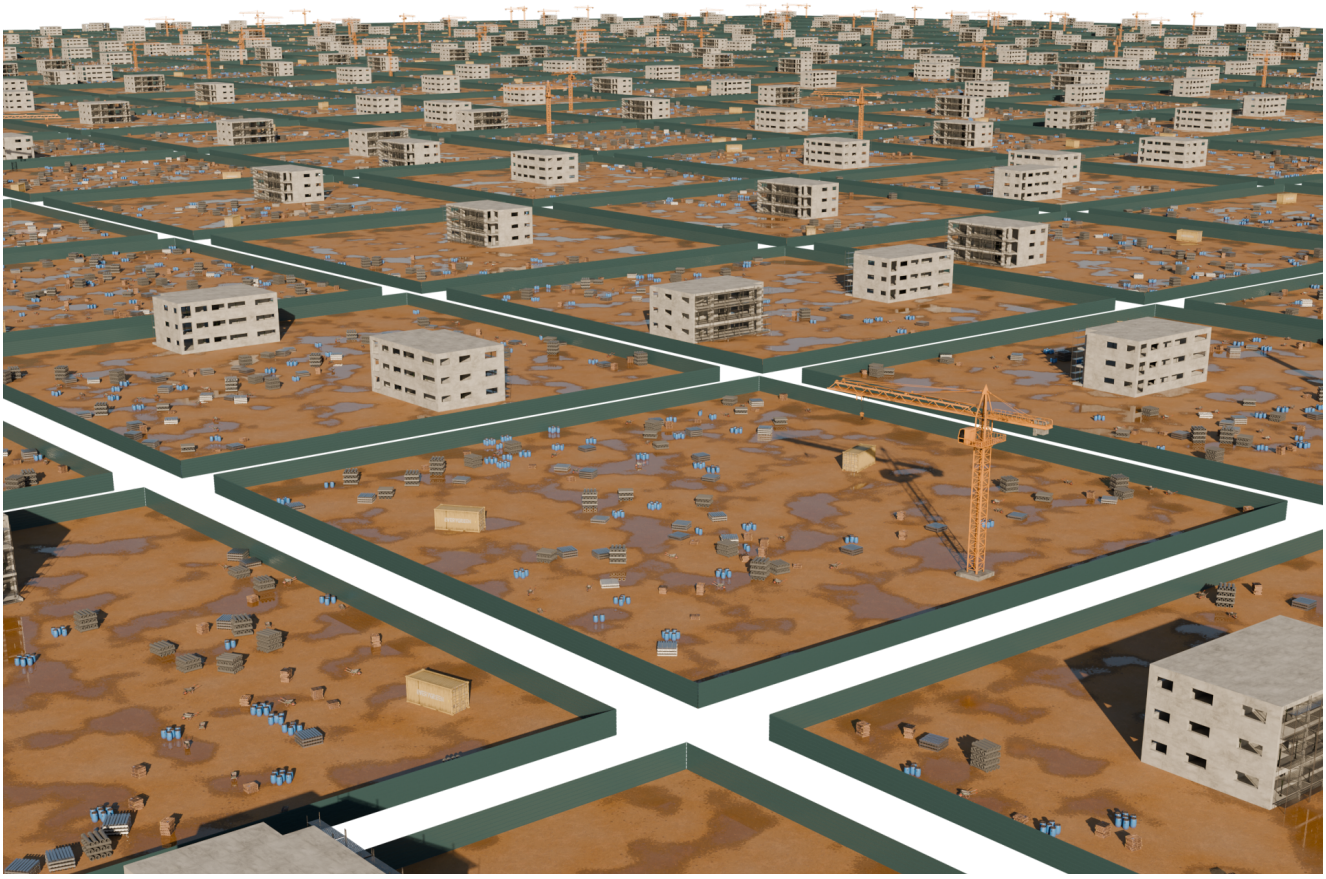


Figure 7. **Scenes generated with our procedural construction site generator.** As this work was driven by the need for a realistic construction site simulator, we also provide a Blender file that can generate random scenes using Geometry Nodes. These scenes consist of a set of 3D objects such as containers, cranes and buildings, that are arranged in a random manner on a flat terrain with random reflection and roughness properties. Object densities, random patterns and scene sizes can be easily adjusted using the Geometry Nodes parameters.

Seed	Random walk AUC ↑	Modified random walk AUC ↑
0	0.22 ± 0.05	0.38 ± 0.02
137	0.26 ± 0.07	0.46 ± 0.04
281	0.19 ± 0.05	0.35 ± 0.07

Table 2. **Performance of exploration baselines.** We report the AUC of the scene coverage evolution during the exploration of 3 generated scenes for reference. We perform 10 runs for each scene and method, and report AUC mean and standard deviation. The modified random walk approach significantly outperforms the random walk method, mostly because its trajectories are less chaotic.

a distance of less than 20 meters to the camera. We evaluate these methods on a set made of 3 scenes generated with our procedural generator, with seeds 0, 137, and 281.

Results. After exploration, we can access the scene's observation history at every step of the agent's trajectory. Similarly to other methods [7, 8], we use this history to compute the AUC of the evolution of the scene coverage during exploration. Table 2 reports the results of our baselines. Figure 8 illustrates one of the trajectory obtained using the modified random walk strategy.

4.3. 3D semantic scene reconstruction

This task addresses the challenge of simultaneously reconstructing the scene's 3D point cloud and performing semantic segmentation on the estimated point cloud [13].

Experimental setup. We implement a baseline exploration strategy using a random walk approach similar to our next-best-view experiment. The agent performs 100 exploration steps, executing the following operations at each step: 1) selecting one of the 5 possible movements as defined in the next-best-view task; 2) acquiring RGB, depth, and semantic segmentation maps; 3) unprojecting the depth map to recover 3D points in the world frame; 4) transforming these points into voxels; 5) assigning semantic labels to

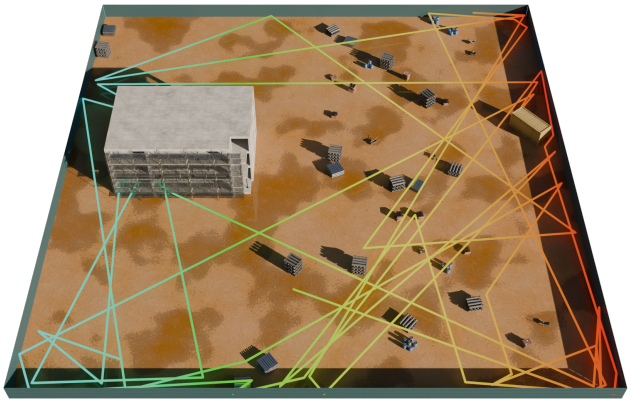


Figure 8. **Modified random walk trajectory.** We show the trajectory computed by our modified random walk strategy for 500 steps on a procedural construction site scene with seed 0. The camera moves forwards until it reaches an obstacle, in which case if performs a random rotation.

mIoU ↑	Chamfer-L1 ↓		Semantic Chamfer ↑	
	Pred	GT	Pred	GT
0.97	0.08	4.56	0.94	0.23

Table 3. **Point cloud reconstruction and semantic segmentation.** We report mIoU for point cloud semantic segmentation, Chamfer-L1 distance in meters for 3D point cloud estimation and Semantic Chamfer score for evaluating both tasks simultaneously.

voxels based on the segmentation map. This process results in a 3D labeled point cloud of the observed portions of the scene. We evaluate this baseline on our procedurally generated construction site environment with a fixed random seed of 0 and a voxel size of 0.05 meter.

Results. Traditional point cloud semantic segmentation [16] only focuses on annotating each point with the correct label. Consequently, mIoU serves as the standard evaluation metric. Separately, the quality of 3D point cloud reconstruction is typically assessed using the Chamfer distance, or Chamfer-L1 distance for more robustness to outliers. We report both of these conventional metrics in Tab. 3. However, our task encompasses both 3D point cloud estimation and semantic annotation simultaneously, necessitating a more comprehensive evaluation approach. To address this challenge, we propose a novel metric, the Semantic Chamfer score, that integrates both geometric proximity and semantic accuracy. For a given pair of point clouds A and B, the Semantic Chamfer score from A to B reports the percentage of points in cloud A that: 1) have a corresponding point in cloud B within a specified distance threshold; and 2) carries the correct semantic label. Similarly to the Chamfer distance, we compute this metric

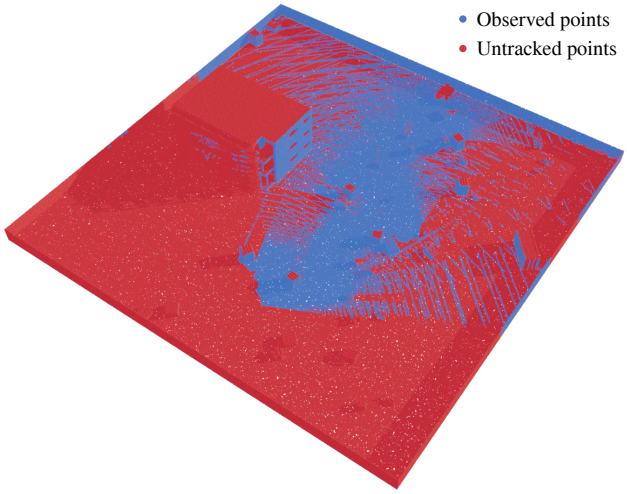


Figure 9. **Chamfer score from ground truth to predicted point cloud.** In blue, we show all points of the ground truth point cloud that have a neighbor in the estimated point cloud and have the correct label. In red, we show the points that did not have any neighbor. Since our baseline uses ground truth semantic segmentation maps, we do not show points that have a neighbor but were labeled incorrectly, as there are only a few of them. The Semantic Chamfer score is the percentage of blue points.

both from the estimated point cloud to the ground truth and vice versa. In our evaluation, using a 0.1 meter distance threshold, the Semantic Chamfer score from prediction to ground truth reaches 0.94, confirming high accuracy in reconstructed regions—an expected outcome given our use of ground truth depth and segmentation maps. Conversely, the ground truth to prediction score of 0.23 indicates that only a quarter of the scene was observed during random exploration. Figure 9 highlights these results and provides insights on how the Chamfer Semantic score is computed.

5. Conclusion

We introduced Buildee, a 3D simulation framework based on Blender designed to evaluate scene exploration, reconstruction, and understanding. By leveraging Blender's powerful rendering engines and flexibility, Buildee allows users to balance realism and simulation speed. Our framework comes with key features that allow embodied agents to navigate the scene and gather data for 3D scene understanding, including RGB-D images and 2D / 3D point tracking. We demonstrated our framework abilities on several tasks, including exploration, point cloud reconstruction and semantic segmentation.

Acknowledgments. This project was funded by the European Union (ERC Advanced Grant explorer, Funding ID 101097259).

References

[1] Carlos E. Agüero, Nate Koenig, Ian Chen, Hugo Boyer, Steven Peters, John Hsu, Brian Gerkey, Steffi Paepcke, Jose L. Rivero, Justin Manzo, Eric Krotkov, and Gill Pratt. Inside the Virtual Robotics Challenge: Simulating Real-Time Robotic Disaster Response. *IEEE Transactions on Automation Science and Engineering*, 12(2):494–506, 2015. 2

[2] Genesis Authors. Genesis: A Universal and Generative Physics Engine for Robotics and Beyond, 2024. 2

[3] Jens Behley, Martin Garbade, Andres Milioto, Jan Quenzel, Sven Behnke, Jürgen Gall, and Cyrill Stachniss. Towards 3D LiDAR-based semantic scene understanding of 3D point cloud sequences: The SemanticKITTI Dataset. *The International Journal of Robotics Research*, 40(8-9):959–967, 2021. 1, 3

[4] Li Chen, Penghao Wu, Kashyap Chitta, Bernhard Jaeger, Andreas Geiger, and Hongyang Li. End-to-End Autonomous Driving: Challenges and Frontiers. *IEEE TPAMI*, 46(12): 10164–10183, 2024. 3

[5] Maximilian Denninger, Martin Sundermeyer, Dominik Winkelbauer, Youssef Zidan, Dmitry Olefir, Mohamad Elbadrawy, Ahsan Lodhi, and Harinandan Katam. BlenderProc. *arXiv preprint*, 2019. 2, 3

[6] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. CARLA: An Open Urban Driving Simulator. In *Proceedings of the 1st Annual Conference on Robot Learning*, pages 1–16, 2017. 1, 2, 3

[7] Antoine Guedon, Pascal Monasse, and Vincent Lepetit. SCONE: Surface Coverage Optimization in Unknown Environments by Volumetric Integration. In *NeurIPS*, pages 20731–20743, 2022. 1, 2, 3, 7

[8] Antoine Guédon, Tom Monnier, Pascal Monasse, and Vincent Lepetit. MACARONS: Mapping and Coverage Anticipation With RGB Online Self-Supervision. In *CVPR*, pages 940–951, 2023. 1, 2, 3, 7

[9] Ziniu Hu, Ahmet Iscen, Aashi Jain, Thomas Kipf, Yisong Yue, David A Ross, Cordelia Schmid, and Alireza Fathi. SceneCraft: An LLM Agent for Synthesizing 3D Scenes as Blender Code. In *ICML*, 2024. 2

[10] Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis. 3D Gaussian Splatting for Real-Time Radiance Field Rendering. *ACM Transactions on Graphics*, 42(4), 2023. 3

[11] Shiyao Li, Antoine Guédon, Clémentin Boittiaux, Shizhe Chen, and Vincent Lepetit. NextBestPath: Efficient 3D Mapping of Unseen Environments. *arXiv preprint*, 2025. 1, 3

[12] Jiayi Liu, Ali Mahdavi-Amiri, and Manolis Savva. PARIS: Part-level Reconstruction and Motion Analysis for Articulated Objects. In *ICCV*, pages 352–363, 2023. 3

[13] Romain Loiseau, Mathieu Aubry, and Loïc Landrieu. Online Segmentation of LiDAR Sequences: Dataset and Algorithm. In *ECCV*, pages 301–317, 2022. 1, 7

[14] Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis. In *ECCV*, pages 405–421, 2020. 3

[15] Xavi Puig, Eric Undersander, Andrew Szot, Mikael Dallaire Cote, Ruslan Partsey, Jimmy Yang, Ruta Desai, Alexander William Clegg, Michal Hlavac, Tiffany Min, Theo Gervet, Vladimír Vondruš, Vincent-Pierre Berges, John Turner, Oleksandr Maksymets, Zsolt Kira, Mrinal Kalakrishnan, Jitendra Malik, Devendra Singh Chaplot, Unnat Jain, Dhruv Batra, Akshara Rai, and Roozbeh Mottaghi. Habitat 3.0: A Co-Habitat for Humans, Avatars and Robots. *arXiv preprint*, 2023. 2, 3

[16] Gilles Puy, Alexandre Boulch, and Renaud Marlet. Using a Waffle Iron for Automotive Point Cloud Semantic Segmentation. In *ICCV*, pages 3379–3389, 2023. 1, 8

[17] Alexander Raistrick, Lahav Lipson, Zeyu Ma, Lingjie Mei, Mingzhe Wang, Yiming Zuo, Karhan Kayan, Hongyu Wen, Beining Han, Yihan Wang, Alejandro Newell, Hei Law, Ankit Goyal, Kaiyu Yang, and Jia Deng. Infinite Photorealistic Worlds Using Procedural Generation. In *CVPR*, pages 12630–12641, 2023. 2

[18] Alexander Raistrick, Lingjie Mei, Karhan Kayan, David Yan, Yiming Zuo, Beining Han, Hongyu Wen, Meenal Parakh, Stamatis Alexandropoulos, Lahav Lipson, Zeyu Ma, and Jia Deng. Infinigen Indoors: Photorealistic Indoor Scenes using Procedural Generation. In *CVPR*, pages 21783–21794, 2024. 2

[19] Nikhila Ravi, Jeremy Reizenstein, David Novotny, Taylor Gordon, Wan-Yen Lo, Justin Johnson, and Georgia Gkioxari. Accelerating 3D Deep Learning with PyTorch3D. *arXiv preprint*, 2020. 2

[20] Jonathan Sauder, Guilhem Banc-Prandi, Anders Meibom, and Devis Tuia. Scalable Semantic 3D Mapping of Coral Reefs with Deep Learning. *arXiv preprint*, 2023. 1

[21] Manolis Savva, Abhishek Kadian, Oleksandr Maksymets, Yili Zhao, Erik Wijmans, Bhavana Jain, Julian Straub, Jia Liu, Vladlen Koltun, Jitendra Malik, Devi Parikh, and Dhruv Batra. Habitat: A Platform for Embodied AI Research. In *ICCV*, 2019. 2, 3

[22] Johannes Lutz Schönberger and Jan-Michael Frahm. Structure-from-Motion Revisited. In *CVPR*, 2016. 1, 3, 5

[23] Andrew Szot, Alexander Clegg, Eric Undersander, Erik Wijmans, Yili Zhao, John Turner, Noah Maestre, Mustafa Mukadam, Devendra Singh Chaplot, Oleksandr Maksymets, Aaron Gokaslan, Vladimír Vondruš, Sameer Dharur, Franziska Meier, Wojciech Galuba, Angel Chang, Zsolt Kira, Vladlen Koltun, Jitendra Malik, Manolis Savva, and Dhruv Batra. Habitat 2.0: Training Home Assistants to Rearrange their Habitat. In *NeurIPS*, pages 251–266, 2021. 2

[24] Shuzhe Wang, Vincent Leroy, Yohann Cabon, Boris Chidlovskii, and Jerome Revaud. DUSt3R: Geometric 3D Vision Made Easy. In *CVPR*, pages 20697–20709, 2024. 1

[25] Fei Xia, Amir R. Zamir, Zhiyang He, Alexander Sax, Jitendra Malik, and Silvio Savarese. Gibson Env: Real-World Perception for Embodied Agents. In *CVPR*, 2018. 2, 3