Sceniris: A Fast Procedural Scene Generation Framework

Anonymous Author(s)
Affiliation
Address
email

Abstract: Synthetic 3D scenes are essential for developing Physical AI and generative models. Existing procedural generation methods often have low output throughput, creating a significant bottleneck in scaling up dataset creation. In this work, we introduce Sceniris, a highly efficient procedural scene generation framework for rapidly generating large-scale, collision-free scene variations. Sceniris also provides an optional robot reachability check for robot tasks. Leveraging batch sampling and faster collision checking in cuRobo, Sceniris achieves 234× higher valid-scene throughput than Scene Synthesizer in the cold start case and 2,936× in the warm start case. Sceniris also expands the object-wise spatial relationships available in prior work to support diverse scene requirements. Our code will be available.

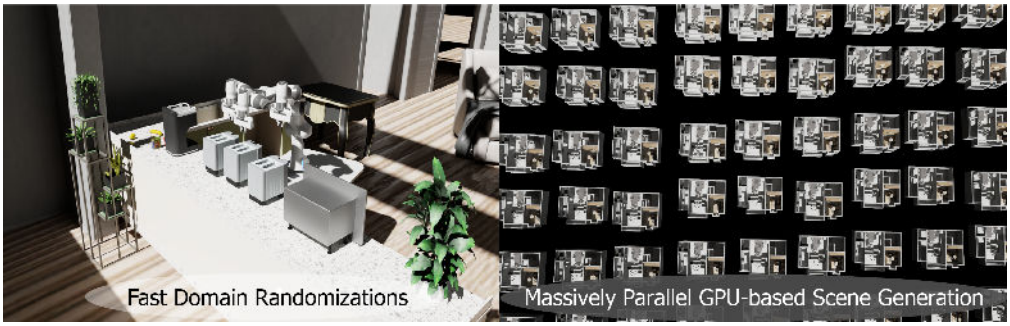


Figure 1: Sceniris is a procedural scene generation tool that provides fast, scalable, and physical-AI plausible scene generation. We leverage GPU to accelerate the collision checking and robot reachability check.

1 INTRODUCTION

Collecting sufficient real-world data is extremely time-consuming [1, 2, 3]. As a result, synthetic environments and data are another important source of data [4, 5, 6, 7, 8].

We aim to generate random object poses that follow the scene layout configuration provided by the user and ensure the scene is collision-free. However, the procedural-based approach often takes a long time to produce a dataset at an acceptable scale [9, 10]. We introduce Sceniris, a fast procedural scene generation framework based on Scene Synthesizer [4]. Specifically, we leverage faster APIs in geometry libraries [11] and use batch matrix computations to speed up sampling, and we use cuRobo [12] to speed up collision checking. In addition, we add more spatial relationships and an optional robot reachability check.

2 RELATED WORK

Randomizing the object poses within a scene is important to improve the generalizability of the policy and increase the diversity of the data [13, 14, 8, 15]. Generative simulations [5, 16, 17, 18, 6],

Submitted to the 10th Conference on Robot Learning (CoRL 2026). Do not distribute.

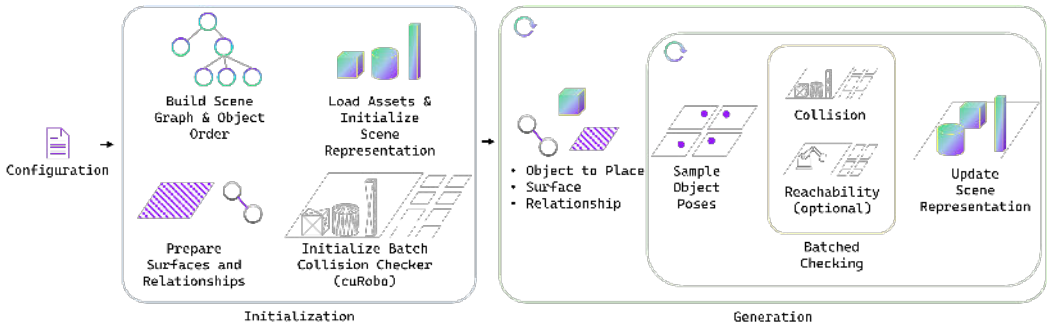


Figure 2: Sceniris takes a configuration and generates a batch of scenes efficiently. Alternatively, Sceniris can generate a scene by adding objects step-by-step, but the efficiency is not optimized.

which generate the scene on-the-fly, require a reliable approach to generate object layouts. Infinigen [19] can generate high-quality multi-room scale indoor scenes using Monte-Carlo Metropolis-Hastings sampling. ProcTHOR [5] also generates indoor scenes and places objects in a collision-free manner. Scene Synthesizer [4] generates object layouts by rejection sampling with collision checking. Table 1 compares procedural scene generation approaches. Learning-based methods, like steerable scene generation [9], also require a huge dataset to train.

3 METHOD

Table 1: Comparison between Sceniris and existing procedural scene generation approaches, including standalone approaches and embedded algorithms in simulation frameworks

Framework	Scene Representation	Sampling	Collision Checking	Batched	Spatial Relationships	Reachability
Standalone						
Sceniris (Ours)	Batched Trimesh	Rejection Sampling	cuRobo	yes	pair and multi-object	✓
Scene Synthesizer [4]	Trimesh	Rejection Sampling	FCL (Trimesh)	✗	'on', object connect	✗
Infinigen-indoors [19]	Trimesh	Monte-Carlo Metropolis–Hastings	✓	✗	✓	✗
Spatial Scene Grammars [10]	Tree	unknown	Drake	✗	✓	✗
In generative simulation						
RoboGen [16]	N/A	Rejection Sampling/Collision Resolving	PyBullet	✗	{'on', 'in'}	✗
GenSim2 [18]	N/A	Predefined + Small-range Sampling	✗	✗	✗	✗
GenSim [17]	N/A	Rejection Sampling	2D occupancy	✗	✗	✗
RoboTwin 2.0 [6]	N/A	Small-range Sampling	✗	✗	✗	✗
In simulation and game engines						
Robocasa [8]	N/A	✗	✗	✗	✗	✗
LIBERO [20]	BDDL	Rejection Sampling	N/A	✗	✗	✗
VIMA [14]	N/A	Rejection Sampling	2D Occupancy	✗	✗	✗
RLBench [13]	CoppeliaSim	Rejection Sampling	Bounding Box	✗	✗	✗
ProcTHOR [5]	Trimesh	Rejection Sampling	FCL (Trimesh)	✗	✓	✗
UE5 PCG [21]	Graph	Rejection sampling	Built-in Tool	✗	✓	✗

Scene Synthesizer [4] samples **one** possible object pose at a time, then checks collisions using the CPU-based Flexible Collision Library (FCL). Our key design idea is to sample a large number of scenes in a batch that leverages the parallelization capability of computing libraries.

3.1 Batched Scene Generation

Sceniris first loads the assets, determines the object placement order, and prepares the pose samplers and collision checker (Figure 2). For each object, Sceniris samples its poses, performs collision checking, and retries failed placements up to a configured limit.

Each scene-graph edge stores N transform matrices as an $(N, 4, 4)$ tensor, representing the spatial relationship between two nodes across the batch. Batched forward kinematics allows us to modify the poses of articulated object parts in parallel.

3.2 Batch Sampling and Caching

The original Scene Synthesizer samples one 2D point from a support polygon at a time. Instead, we first sample a larger batch than required and then cache the batch in a queue. When the cached samples run out, it samples another batch. When support polygons share the same surface with different world poses, we sample from one canonicalized polygon and transform each point using its corresponding surface pose. For complex spatial constraints that cannot be affine-transformed from the first scene instance (Section 3.4), the system falls back to sample points from each polygon.

3.3 Collision and Reachability Checks

We use cuRobo [12] to perform collision checking for a batch of scenes on GPU. We pre-cache all meshes and world configurations during initialization, then only modify mesh enabled/disabled states and transforms. During retry, collision checking is performed only on scenes that require retry. We also cache the asset-level Trimesh scene to avoid repeatedly converting object assets.

We utilize RM4D [22], a reachability map that uses a single 4D data structure to support both forward and inverse queries. We batch process queries and transfer the data structure to the GPU. Given a robot position and selected objects or object parts, this optional checker rejects sampled poses that are not reachable by the robot.

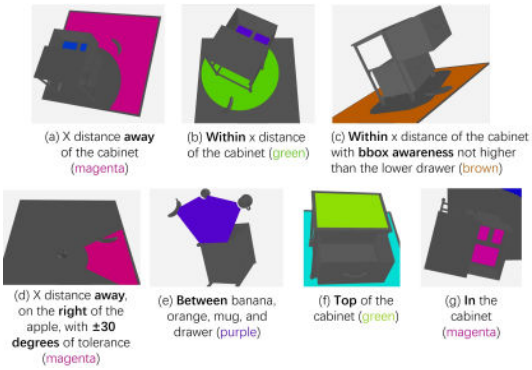


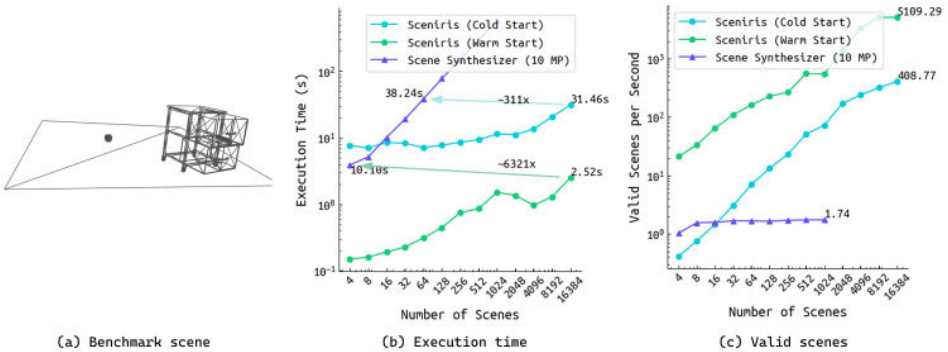
Figure 3: Examples of additional spatial relationships supported by Sceniris

3.4 Extending Spatial Relationships

We extend Scene Synthesizer's random and side-by-side placements to support more spatial relationships (Figure 3).

Object-parent relationship Sceniris supports `'on'` and `'inside'`, selecting parent mesh surfaces without or with a "roof", respectively.

Object-object relationships Our schema specifies anchor objects $\mathcal{O}$, direction $\mathbf{v}$, distance d, and distance type d_t. For one anchor, $\mathbf{v}$ specifies left, right, front, back, an arbitrary direction vector, or no directional restriction, in a global or local reference frame. The distance types `'greater'`, `'less'`, and `'equal'` place an object at least d, at most d, or approximately d from the anchor. We construct a polygonal approximation of a (partial) annulus S_r using the direction, distance, and angular tolerance, then sample from $S' = S_r \cap S$, where S is the original support polygon. The `face_to` attribute forces an object to face another object. For two or more anchors, $d_t =$ 'middle' places the object in the space between the anchors.



(a) Benchmark scene (b) Execution time (c) Valid scenes

Figure 4: (a) The benchmark scene, simplified for visualization. (b) Execution time. (c) The number of valid environments generated per second. Sceniris handles 311 and 6,321 times more environments per second than 10 multi-processed Scene Synthesizer, under cold start and warm start, respectively.

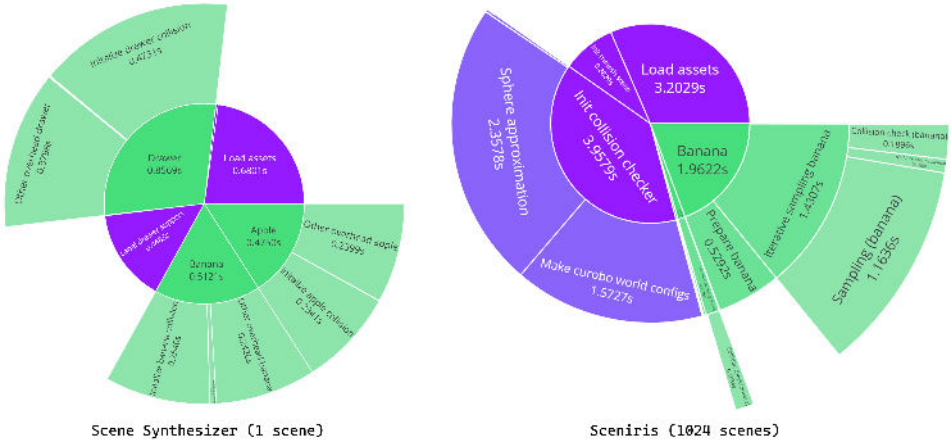


Figure 5: Total time spent on key steps in Scene Synthesizer (**1** scene) and Sceniris (ours, **1024** scenes). Both approaches use the same scene configuration in Section 4.1. Regions in green show the time spent on placing objects.

Object-support relationship The `ratio_on_support` parameter loosely controls surface-boundary placement: 1 ideally keeps the entire object projection supported, while 0 requires only its pivot on the surface. We erode the support polygon by this ratio times half the shorter edge of the projected object mesh. This approximation does not guarantee the exact configured ratio.

4 EXPERIMENTS

4.1 Batched Scene Generation

We benchmark a scene (Figure 4(a)) with an apple and a cabinet randomly placed on a plane, a banana inside the upper drawer, and randomized drawer joints. All spatial relationships are supported by both methods. The benchmark is conducted on a Google Cloud VM with 16 CPU threads (Intel Xeon CPU @ 2.20GHz) and an NVIDIA L4 GPU. We run Scene Synthesizer on 10 processes simultaneously, evenly distributing the scene instances. Each object is allowed 10 collision retries; unresolved scenes are marked invalid.

We vary the total number of scenes to generate from 4 to 16,384. Sceniris processes 16,384 scenes in 32s, while Scene Synthesizer processes 64 scenes in 39s, giving about 311× higher throughput. A warm start executes only the "Generation" step in Figure 2, finishing 16,384 scenes in 2.52s. Considering only valid scenes, Sceniris achieves 234× higher throughput in the cold start case and 2,936× in the warm start case (Figure 4(c)). This supports continuous scene randomization for parallel reinforcement learning or data collection.

4.2 Scaling up with Complex Spatial Relationships

We extend the benchmark configuration (Mid) to Hard and Hard+. Hard places the apple at least 0.5m in front of the drawer and an orange between them. Hard+ adds a mug to the drawer's right within 0.7m, with the orange between the drawer, mug, and apple. Hard and Hard+ use two and three complex spatial relationships, respectively. Scene Synthesizer does not support these relationships, preventing a direct comparison. Each level adds about 20–30s for cold starts and 20s for warm starts (Figure 6), because some constraints require per-scene sampling polygons rather than a shared rigid transformation. Nevertheless, Sceniris still achieves about 249 valid scenes/s in Hard+ (cold start). Batch sampling from heterogeneous polygons on a GPU is a potential future improvement.

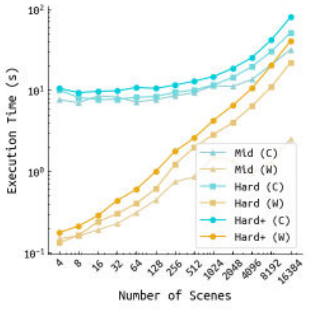


Figure 6: Execution times for Mid, Hard (2 complex relationships), and Hard+ (3). C: cold start; W: warm start.

4.3 Component Breakdown

According to Figure 5, Sceniris spends 10.17s on generating **1,024** scenes and Scene Synthesizer takes 2.99s for **1** scene. Much of the baseline's "other overhead" comes from repeatedly converting object assets to Trimesh scenes, which Sceniris caches. Enabling reachability checks for the apple and cabinet has little effect on runtime: each query takes ∼0.0001s, at most 1/20 of the collision check cost. Initializing cuRobo world configurations remains a significant overhead and an opportunity for future improvement.

Pressure test A batch of 524,288 scenes fits in an L4 GPU using about 20GB VRAM. It takes 769.67s from a cold start, with 113,820 invalid scenes, yielding 533.30 valid scenes/s versus 408.77 in the smaller benchmark. Warm start throughput reaches 5,279.84 valid scenes/s.

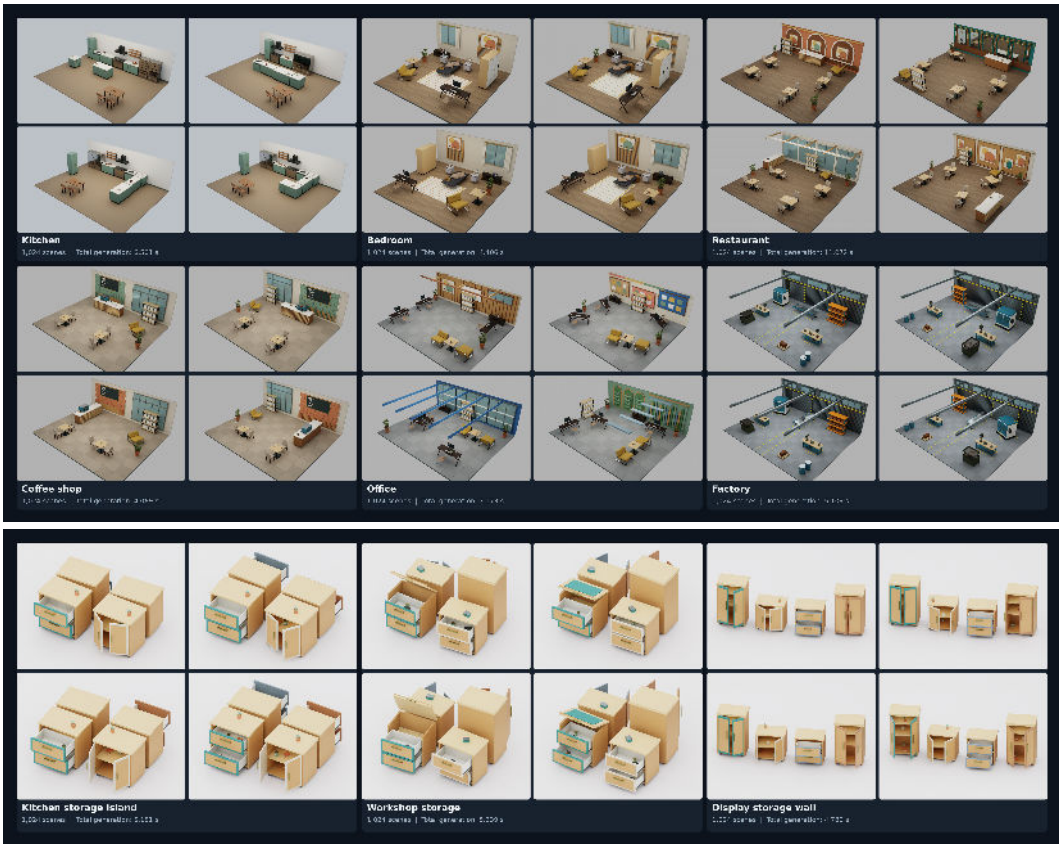


Figure 7: Generated scenes: six room-scale configurations (top) and three articulated-object configurations (bottom). Each group shows four randomized instances and the time to generate 1,024 scenes (on RTX6000 Ada).

Figure 7 shows room-scale and articulated-object variations with batch generation times.

5 CONCLUSIONS

Sceniris uses batched sampling, caching, and GPU collision checking to generate collision-free scenes with rich spatial relationships and optional robot reachability checks. It achieves 234× higher valid-scene throughput than multi-process Scene Synthesizer from a cold start and 2,936× from a warm start, supporting large-scale scene randomization.

References

[1] A. Khazatsky, K. Pertsch, S. Nair, A. Balakrishna, S. Dasari, S. Karamcheti, S. Nasiriany, M. K. Srirama, L. Y. Chen, K. Ellis, et al. Droid: A large-scale in-the-wild robot manipulation dataset. *arXiv preprint arXiv:2403.12945*, 2024.

[2] K. Black, N. Brown, D. Driess, A. Esmail, M. Equi, C. Finn, N. Fusai, L. Groom, K. Hausman, B. Ichter, et al. pi0: A vision-language-action flow model for general robot control. *arXiv preprint arXiv:2410.24164*, 2024.

[3] G. Team. Galaxea g0: Open-world dataset and dual-system vla model. *arXiv preprint arXiv:2509.00576v1*, 2025.

[4] C. Eppner, A. Murali, C. Garrett, R. O'Flaherty, T. Hermans, W. Yang, and D. Fox. scene_synthesizer: A python library for procedural scene generation in robot manipulation. *Journal of Open Source Software*, 2024.

[5] M. Deitke, E. VanderBilt, A. Herrasti, L. Weihs, J. Salvador, K. Ehsani, W. Han, E. Kolve, A. Farhadi, A. Kembhavi, and R. Mottaghi. ProcTHOR: Large-Scale Embodied AI Using Procedural Generation. In *NeurIPS*, 2022. Outstanding Paper Award.

[6] T. Chen, Z. Chen, B. Chen, Z. Cai, Y. Liu, Q. Liang, Z. Li, X. Lin, Y. Ge, Z. Gu, et al. Robotwin 2.0: A scalable data generator and benchmark with strong domain randomization for robust bimanual robotic manipulation. *arXiv preprint arXiv:2506.18088*, 2025.

[7] A. Mandlekar, D. Xu, J. Wong, S. Nasiriany, C. Wang, R. Kulkarni, L. Fei-Fei, S. Savarese, Y. Zhu, and R. Martín-Martín. What matters in learning from offline human demonstrations for robot manipulation. In *Conference on Robot Learning (CoRL)*, 2021.

[8] S. Nasiriany, A. Maddukuri, L. Zhang, A. Parikh, A. Lo, A. Joshi, A. Mandlekar, and Y. Zhu. Robocasa: Large-scale simulation of everyday tasks for generalist robots. In *Robotics: Science and Systems*, 2024.

[9] N. Pfaff, H. Dai, S. Zakharov, S. Iwase, and R. Tedrake. Steerable scene generation with post training and inference-time search, 2025. URL https://arxiv.org/abs/2505.04831.

[10] G. Izatt and R. Tedrake. Generative modeling of environments with scene grammars and variational inference. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6891–6897. IEEE, 2020.

[11] S. Gillies et al. Shapely: manipulation and analysis of geometric objects, 2007–. URL https://github.com/Toblerity/Shapely.

[12] B. Sundaralingam, S. K. S. Hari, A. Fishman, C. Garrett, K. V. Wyk, V. Blukis, A. Millane, H. Oleynikova, A. Handa, F. Ramos, N. Ratliff, and D. Fox. curobo: Parallelized collision-free minimum-jerk robot motion generation, 2023.

[13] S. James, Z. Ma, D. Rovick Arrojo, and A. J. Davison. Rlbench: The robot learning benchmark & learning environment. *IEEE Robotics and Automation Letters*, 2020.

[14] Y. Jiang, A. Gupta, Z. Zhang, G. Wang, Y. Dou, Y. Chen, L. Fei-Fei, A. Anandkumar, Y. Zhu, and L. Fan. Vima: General robot manipulation with multimodal prompts. In *Fortieth International Conference on Machine Learning*, 2023.

[15] R. Gong, J. Huang, Y. Zhao, H. Geng, X. Gao, Q. Wu, W. Ai, Z. Zhou, D. Terzopoulos, S.-C. Zhu, et al. Arnold: A benchmark for language-grounded task learning with continuous states in realistic 3d scenes. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2023.

[16] Y. Wang, Z. Xian, F. Chen, T.-H. Wang, Y. Wang, K. Fragkiadaki, Z. Erickson, D. Held, and C. Gan. Robogen: Towards unleashing infinite data for automated robot learning via generative simulation. *arXiv preprint arXiv:2311.01455*, 2023.

[17] L. Wang, Y. Ling, Z. Yuan, M. Shridhar, C. Bao, Y. Qin, B. Wang, H. Xu, and X. Wang. Gensim: Generating robotic simulation tasks via large language models. In *Arxiv*, 2023.

[18] P. Hua, M. Liu, A. Macaluso, Y. Lin, W. Zhang, H. Xu, and L. Wang. Gensim2: Scaling robot data generation with multi-modal and reasoning llms, 2024. URL https://arxiv.org/abs/2410.03645.

[19] A. Raistrick, L. Mei, K. Kayan, D. Yan, Y. Zuo, B. Han, H. Wen, M. Parakh, S. Alexandropoulos, L. Lipson, et al. Infinigen indoors: Photorealistic indoor scenes using procedural generation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 21783–21794, 2024.

[20] B. Liu, Y. Zhu, C. Gao, Y. Feng, Q. Liu, Y. Zhu, and P. Stone. Libero: Benchmarking knowledge transfer for lifelong robot learning. *arXiv preprint arXiv:2306.03310*, 2023.

[21] Unreal engine procedural content generation. URL https://dev.epicgames.com/documentation/en-us/unreal-engine/procedural-content-generation-overview.

[22] M. Rudorfer. Rm4d: A combined reachability and inverse reachability map for common 6-/7-axis robot arms by dimensionality reduction to 4d. *arXiv preprint arXiv:2410.06968*, 2024.