arXiv:2512.16896v1 [cs.RO] 18 Dec 2025

Sceniris: A Fast Procedural Scene Generation Framework

Jinghuan Shang[1], Harsh Patel[2], Ran Gong[1], Karl Schmeckpeper[1]

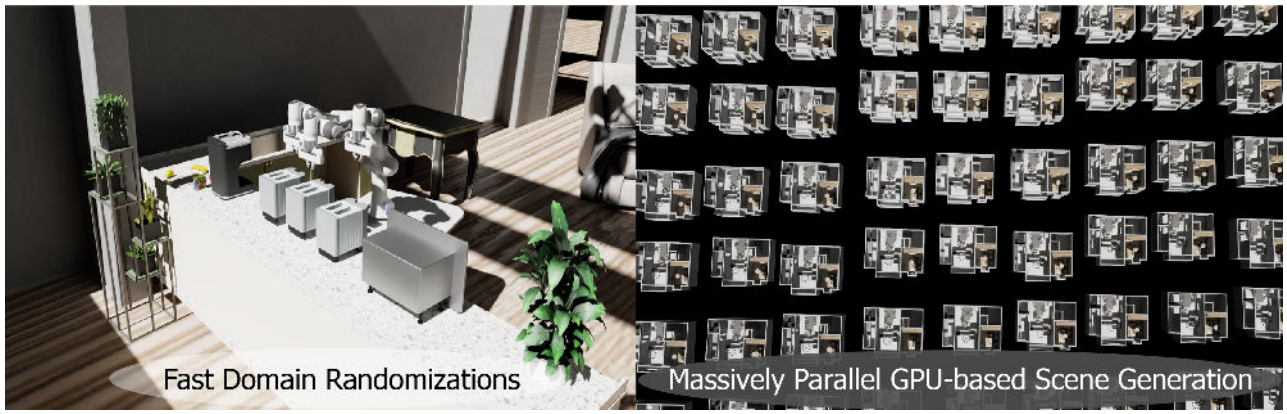

Fig. 1: Sceniris is a procedural scene generation tool that provides fast, scalable, and physical-AI plausible scene generation. We leverage GPU to accelerate the collision checking and robot reachability check.

***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, providing manipulation-feasible scenes for robot tasks. Sceniris is designed for maximum efficiency by addressing the primary performance limitations of the prior method, Scene Synthesizer. Leveraging batch sampling and faster collision checking in cuRobo, Sceniris achieves at least 234x speed-up over Scene Synthesizer. Sceniris also expands the object-wise spatial relationships available in prior work to support diverse scene requirements. Our code is available at `https://github.com/rai-inst/sceniris`**

I. INTRODUCTION

Data-driven approaches are one of the most effective methods to develop AI when data are sufficient. Internet-scale language and image data have successfully spawned many Large Language Models (LLMs) [1]–[4], Vision Foundation Models (VFMs) [5]–[10], and Vision-Language Models (VLMs) [11]–[15]. Moving forward from word and pixels, the community is developing AI that perceives, generates copies of, or interacts with the physical world where the existing dataset is limited [16]–[20]. Collecting sufficient real-world data is also extremely time-consuming [18], [21], [22]. As a result, synthetic environments and data are another important source of data [23]–[27].

In this work, we focus on providing synthetic scenes for physical AI, like simulated environments for robots [27]–[30], training generative AI [31], [32], and 3D perception tasks [33]–[35] at a large scale. We aim to generate random object poses that follow the scene layout configuration provided by the user and ensure the scene is collision-free. Existing works on generating synthetic scenes are either procedural-based [23], [24], [36] or learning-based [32], [37], [38]. Learning-based methods, which are mostly generative AI, require a dataset that is usually generated by procedural-based methods. However, the procedural-based approach often takes a long time to produce a dataset at an acceptable scale [31], [36]. The efficiency of the procedural-based approach becomes a key bottleneck for scaling up.

We introduce Sceniris, a fast procedural scene generation framework that generates a large number of collision-free variations of scenes in a very short time. Sceniris also provides robot reachability check for cases generating scenes that guarantee specified objects are reachable by the robot, which is essential for embodied AI, and none of the existing scene generation framework supports this. Sceniris is designed with the top priority on efficiency. We propose the idea of parallelizing the sampling process and collision checking, two major bottlenecks in the existing procedural generation method, to achieve the best efficiency. We demonstrate our parallelization based on a prior work on procedural scene generation, Scene Synthesizer [23]. Specifically, we leverage faster APIs in geometry libraries [39] and use batch matrix computations to speed up sampling, and we use cuRobo [40] to speed up collision checking. In addition, we add more spatial relationships that are not available in scene synthesizer [23]. With all the improvements, Sceniris achieves more than 234× speed-up than scene synthesizer [23], showing its strong ability for scaling up the procedural scene generation.

[1]Robotics and AI Institute, Boston, MA, USA. {jshang, rgong, kschmeckpeper}@rai-inst.com

[2]University of Waterloo, Ontario, Canada. Work done during the internship at Robotics and AI Institute. h329pate@uwaterloo.ca

II. RELATED WORK

A. Applications that Require 3D Scenes

The demand for diverse 3D scenes has grown significantly across various domains. In robotics, diverse scenes are required in simulation [41]–[52] to train policies [27], [53]–[59] and collect data [26]. Randomizing the object poses within a scene is also important to improve the generalizability of the policy and increase the diversity of the data [27], [53], [56], [58]. Moreover, digital twins [60] and digital cousins [61] aim at minimizing sim-to-real gap, where higher quality 3D scenes are more demanded. Generative simulations [24], [25], [28]–[30], which generate the scene on-the-fly, require a reliable approach to generate object layouts. Training generative models for 3D scenes also requires a huge amount of data [31], [32], [37], [38], [62]–[65], where procedure scene generation [23], [36], [66]–[69] methods are often required to produce them. 3D perception tasks can also benefit from a large 3D scene dataset. Therefore, we believe procedural scene generation methods are a good fit for all these applications, and the methods that have higher output throughput will greatly support the high data demand.

B. Scene Generation Approaches

Procedural-based and learning-based scene generation approaches are the two ways to obtain synthesized scenes for the application above. Procedural-based methods often involve sampling procedures and require a schema or a configuration to decide the scene layout. Infinigen [67], [68] can generate high-quality multi-room scale indoor scenes using Monte-Carlo Metropolis-Hastings sampling. ProcTHOR [24] also generates indoor scenes and places objects in a collision-free manner. Scene Synthesizer [23] can compose both procedural-based objects and scenes like kitchens, and the object layout is generated by rejection sampling with collision checking. Spatial scene grammars [36] generate the scenes on CPU. There are also algorithms embedded in simulation frameworks like GenSim [28], GenSim2 [29], RoboGen [30], and Robocasa [27], to compose the scene and randomize the object poses, but usually the range of randomizing the poses is limited. UE5 PCG [70] plugin is a scene generation tool in the game engine to generate a random scene. A more comprehensive comparison across procedural-based scene generation is available in Table I.

However, the existing methods do not scale to generating a large batch of scenes in a short time. For example, Infinigen takes up to 10 minutes to generate a living room, and Scene Synthesizer generates one table-top scene in about 10 seconds. Under this throughput, generating a large-scale dataset for 3D scenes or performing object pose randomization is difficult. Learning-based methods, like steerable scene generation [31], may also have a similar throughput and require a huge dataset to train. To this end, our objective is to generate 100-1000 scenes within a second on average, providing strong support for data-hungry scenarios.

III. PRELIMINARY: SCENE SYNTHESIZER

Scene Synthesizer [23] is a single-threaded scene generation framework. It procedurally places objects in the scene, as well as generates some kitchen or office objects. In this work, we mainly discuss the object placement part, which is representative of existing single-threaded procedural scene generation methods. Our objective is to design a faster procedural scene generation framework by parallelizing the key steps, and Scene Synthesizer [23] is a great base approach that can demonstrate our improvements. Below, we go through the Scene Synthesizer's scene generation process and key components, and identify components to improve.

a) Main algorithm: Scene Synthesizer [23] uses rejection sampling as the main generation logic. When adding a new object to the scene, it first samples **one** possible pose for the object. The position consists of a 2D position sampled in a previously marked 2D surface plus a collision-avoiding z coordinate, or a 3D position directly sampled from a 3D volume. The orientation is sampled from pre-defined rules (e.g., random rotation along the z-axis) or stable poses computed from the object mesh. Then, the collision checker verifies whether the object can be placed at the sampled pose without any collisions. Retry on an object happens if any collision is detected. The entire main logic runs on a single main thread. We observe an opportunity to speed up the algorithm by parallelization.

b) Scene representation: Scene Synthesizer uses Trimesh to represent the scene as a scene graph, where a node is an object or an object part, and a directed edge is the transform from the source node to the destination node. The transform is stored in the 4×4 homogeneous matrix. As a result, Scene Synthesizer can represent and work on one scene instance at any time. We observe that there is room for improvement through the representation of a batch of scenes.

c) Collision check: Scene Synthesizer uses trimesh's collision manager to detect collision. Trimesh uses Flexible Collision Library (FCL) as the underlying algorithm. FCL runs on the CPU entirely.

d) Position sampling: For an object to be placed on a surface, which is the most common case in scene generation, Scene Synthesizer uses `shapely` [39] to sample one 2D position from the pre-computed surface represented by a polygon. The polygon is called the support polygon. It first randomly samples a 2D point within the polygon and attaches a non-zero offset on the z-axis to avoid collision. Though a vectorized (i.e., high efficiency) API is used during sampling, the actual use is sampling only one position from the polygon at a time, which is not efficient.

IV. METHOD

Sceniris is developed based on Scene Synthesizer [23], and our top priority is improving its efficiency. Our key design idea is to sample a large number of scenes in a batch that leverages the parallelization capability of computing libraries. The main bottlenecks in Scene Synthesizer are pose sampling and collision checking, as we discussed above. To this end, we improve the sampling by batch and improve the collision checking using cuRobo. In addition, to support more spatial relationships for procedural generation, we extend the existing spatial relationships and sample them

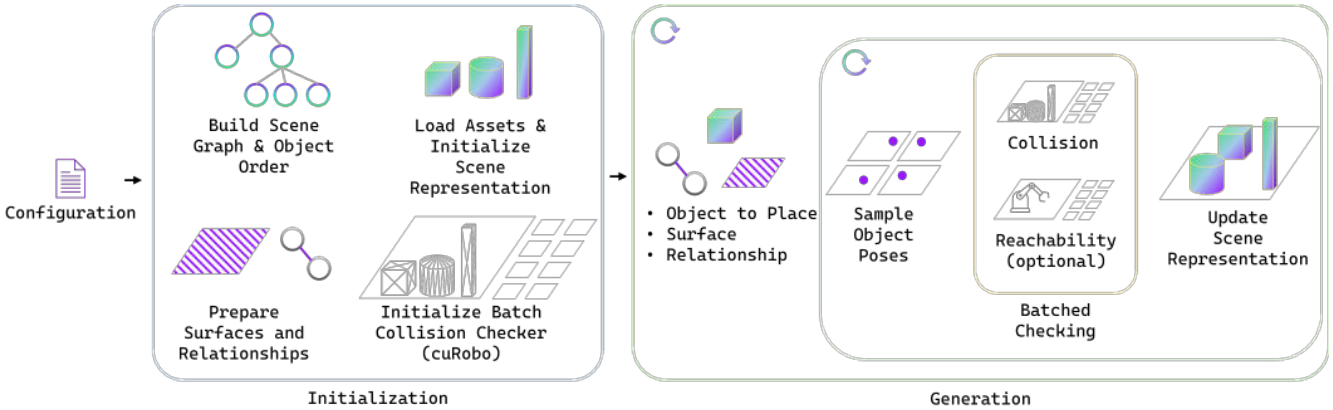

Fig. 2: Sceniris takes a configuration and generates a batch of scenes efficiently. Alternatively, Sceniris can generate a scene by adding objects step-by-step, but the efficiency is not optimized.

TABLE I: 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 [23]	Trimesh	Rejection Sampling	FCL (Trimesh)	✗	'on', object connect	✗
Infinigen-indoors [68]	Trimesh	Monte-Carlo Metropolis–Hastings	✓	✗	✓	✗
Spatial Scene Grammars [36]	Tree	unknown	Drake	✗	✓	✗
In generative simulation						
RoboGen [30]	N/A	Rejection Sampling/Collision Resolving	PyBullet	✗	{'on', 'in'}	✗
GenSim2 [29]	N/A	Predefined + Small-range Sampling	✗	✗	✗	✗
GenSim [28]	N/A	Rejection Sampling	2D occupancy	✗	✗	✗
RoboTwin 2.0 [25]	N/A	Small-range Sampling	✗	✗	✗	✗
In simulation and game engines						
Robocasa [27]	N/A	✗	✗	✗	✗	✗
LIBERO [54]	BDDL	Rejection Sampling	N/A	✗	✗	✗
VIMA [56]	N/A	Rejection Sampling	2D Occupancy	✗	✗	✗
RLBench [53]	CoppeliaSim	Rejection Sampling	Bounding Box	✗	✗	✗
ProcTHOR [24]	Trimesh	Rejection Sampling	FCL (Trimesh)	✗	✓	✗
UE5 PCG [70]	Graph	Rejection sampling	Built-in Tool	✗	✓	✗

efficiently. We also add a reachability feature to support robot manipulation scene generation. We describe these main improvements in the following subsections.

A. System Overview

The main scene generation procedure is shown in Figure 2. Sceniris first initializes the system, including loading the assets, determining the order of object placement, preparing the pose samplers (including spatial relationships), and preparing the collision checker. Then, each object will be placed into the scene. For each object, Sceniris samples its poses, performs collision checking, and optionally retries limited times for environments that fail the check. The scene generation procedure ends once all objects are successfully placed or after the maximum number of retries.

B. Batched Scene Representation

The core supporting modification is batched scene representation – storing a batch of scene instances in the scene graph. We store a batch of N transform matrices in edges that represent the N instances of the spatial relationship between two nodes. For example, if we are generating 8 different scenes where an apple is randomly placed on the table, the (table, apple) edge stores an $(8, 4, 4)$ tensor representing the 8 transform matrices. Based on this batched scene representation, we also implemented a batched forward kinematics function, allowing us to modify the poses of articulated object parts in parallel. With the batched scene representation, Sceniris can support more improvements that require batch operations.

C. Batch Sampling and Caching

The original Scene Synthesizer heavily relies on sampling 2D points from a polygon, but only one point is sampled at a time. We sample a batch of points at a time instead. However, calling sampling multiple times still has overhead. Instead of sampling in an ad-hoc way, 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. With caching, the system can continuously generate more batches of scene instances with less overhead.

The batch sampling method above covers the case of one support polygon. Considering the case that we want to randomly place an apple on the table in N scene instances, and the table is also randomly placed on the ground, the support polygons for the apple are the same table surface with different world poses. For a case like this, instead of sampling points from each polygon, we sample N (or more using cache) points from the canonicalized polygon (usually the first scene instance), and transform each point using the world pose of its corresponding support polygon for each scene instance. This approach significantly reduces overhead for iterating through scene instances. However, it is not always the case that the support polygon can be affine-transformed from the first scene instance. This usually

happens when we customize complex spatial constraints introduced in Section IV-F. For these cases, the system falls back to sample points from each polygon.

D. Collision Checking

Once a batch of poses is sampled, they should be verified by the collision checker to filter out non-collision-free placements. We use cuRobo [40] to perform collision checking for a batch of scenes on GPU. To minimize the overhead, we pre-cached all the meshes and the world configurations that cuRobo requires during the initialization stage, which will be executed only once. We only modify the meshes enabled/disabled state and the transforms through cuRobo's collision managing instance. During retry, the collision checking is performed only on those scenes that require retry, further reducing the overhead compared to checking all the scenes.

E. Reachability

Understanding a manipulator's workspace is essential for many tasks, as it plays a key role in configuring a scene for feasible task execution. To address this, we utilize RM4D [71] — a reachability map that uses a single 4D data structure to support both forward and inverse queries without sacrificing accuracy. To further enhance performance, we batch process queries and transfer the data structure to the GPU, significantly increasing throughput and efficiency. The reachability check feature can be optionally turned on for objects or object parts, ensuring the object or parts are reachable. Given a (hypothesized) robot position in the scene configuration and the objects (parts) that require the reachability check, the checker rejects the sampled object pose if it is not reachable by the robot.



Fig. 3: Examples of additional spatial relationships supported by Sceniris

F. Extending Spatial Relationships

Scene synthesizer [23] supports placing objects randomly or placing an object side-by-side with another object (called `connect`). We extended the framework to support more spatial relationships. Figure 3 shows some spatial relationship examples.

a) Object-parent object spatial relationship: The object-parent object spatial relationship is used to determine the object's base support surface. Sceniris supports two types of spatial relationships: `'on'` and `'inside'`. `'on'` will find parent object mesh surfaces without a "roof", and `'inside'` will find those surfaces with a "roof" to ensure that the object is placed inside the object. We implement these using Scene Synthesizer's original `labal_support` function, and we add an option to fully exclude those surfaces without a "roof" to enable the `'inside'` case. Examples can be found in Figure 3(f) and (g).

b) Object-object spatial relationships: Our schema of object-object spatial relationships can be mainly defined by the following parameters: `anchor object list` $\mathcal{O}$, `direction` $\mathbf{v}$, `distance` d, and `distance_type` d_t. The anchor object list describes all the objects $\mathcal{O} = \{O_1^a, \ldots, O_n^a\}$ that will be considered in the spatial relationship with the object to be added O^t. n is the number of the anchor objects.

One anchor object ($n = 1$). If there is only one anchor object, it describes the spatial relationship between O^a and O_t. Under this scenario, $\mathbf{v}$ is available for $\{-x, x, -y, y, \text{Null}\}$, or equivalently, $\{left, right, front, back, \text{Null}\}$, following the original definition of Scene Synthesizer. We also provide an option to transform these pre-defined directions using a global or local reference frame. When $\mathbf{v} = Null$, O^t can be placed at all directions w.r.t. O^a. $\mathbf{v}$ can also take a direction vector. In addition to $\mathbf{v}$, d is also available for the one anchor object case. Together with d_t, the `distance_type`, $\mathbf{d}$ decides how far O_t should be away from O_a. d_t has three options in this case, {'greater', 'less', 'equal'}, for placing object at least d, at most d, or (softly) exact d far from the anchor. To represent this spatial relationship, we construct a (partial) annulus S_r (approximated by a polygon) using given $\mathbf{v}$ and d_t, and the points inside the annulus are considered to satisfy the condition. In detail, there will be another parameter `direction_angle_threshold`, θ, that controls the maximum angle that deviates from $\mathbf{v}$. Formally, the four key vertices constructing the annulus are $\{p(O_a) + \text{Rot}(\mathbf{v}, \pm\theta) * \text{min_r}, p(O_a) + \text{Rot}(\mathbf{v}, \pm\theta) * \text{max_r}\}$, where $p(\cdot)$ represents the 2D position of the object on the support plane, $\text{Rot}(\cdot, \theta)$ is rotating the direction vector by θ angle on the 2D plane, and min_r, *max_r* are determined by d_t, such that

$$\text{min_r} = \begin{cases} 0 & \text{if } d_t \text{ is 'less'} \\ d & \text{if } d_t \text{ is 'greater'} \end{cases} \quad \text{max_r} = \begin{cases} d & \text{if } d_t \text{ is 'less'} \\ +\text{inf} & \text{if } d_t \text{ is 'greater'} \end{cases}.$$

We then update the final polygon $S' = S_r \cap S$ that object 2D positions are sampled from, where S is the original complete support plane. Examples of single-anchor relationships can be found in Figure 3(a,b,c,d).

In addition to position relationships, we also provide a '`face_to`' attribute to force an object's orientation to face towards another object.

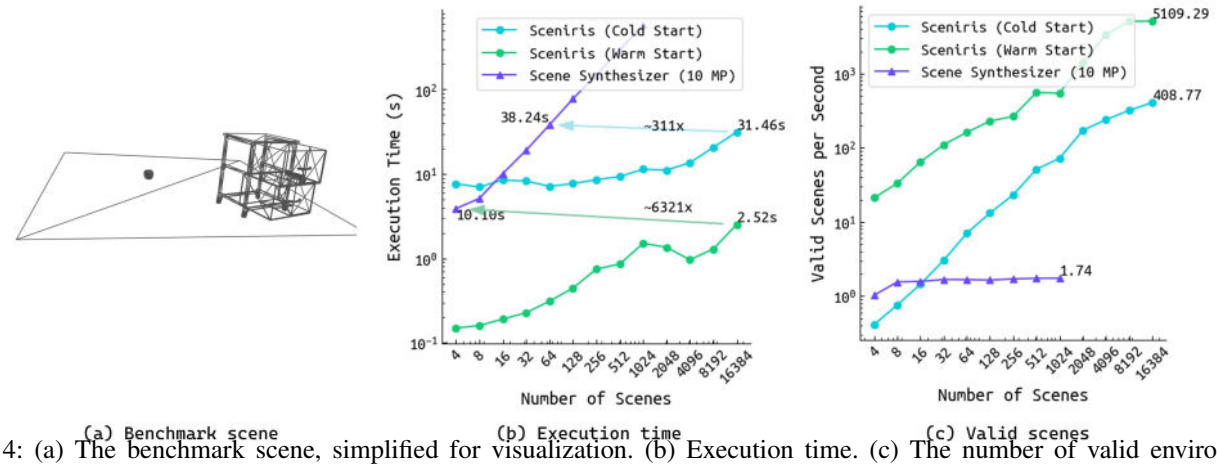

Fig. 4: (a) The benchmark scene, simplified for visualization. (b) Execution time. (c) The number of valid environments generated per second. Sceniris handles 311 and 6,321 times more environments per second than 10 multi-processed Scene Synthesizer, under cold start and warm start, respectively.

Two or more anchor objects ($n \geq 2$). If there are two or more anchor objects, the pair-wise object relationships could be ambiguous. Therefore, we only provide one type of spatial relationship, d_t = 'middle', meaning that the O^t should be placed in the space surrounded by $\mathcal{O}$. We construct a polygon where the vertices of the polygon are the positions of anchor objects $\{p(O_1^a), ...p(O_n^a)\}$, which is used to sample points for object placement. An example of this relationship can be found in Figure 3(e)

c) Object-support surface relationship: We add a configuration term 'ratio_on_support' to loosely control how the object should be placed concerning the surface boundary. When the 'ratio_on_support' is set to 1.0, the surface should ideally hold the entire object's projection, i.e., $P(\text{obj}) \subset$ surface, where $P(\cdot)$ is the projection of the object mesh to the support plane. When the value is set to 0, it is a valid placement as long as the pivot (usually the center of the bottom surface of the object's bounding box) is on the surface. To ensure efficiency, we implement this through estimation ahead of sampling the actual poses of the objects. We use the ratio_on_support $\times$ half length of the shorter edge of the projected object mesh to erode the support surface. So there will be cases where the object does not exactly follow the configured ratio_on_support value.

d) Other improvements: We improved other engineering aspects of the original Scene Synthesizer [23]. We cache the asset-level Trimesh scene, the result of `asset.as_trimesh_scene()`, because we find the operation is time-consuming, and it is used in a lot of places. We modified the visualization so that the generated batch of scenes can be visualized in the same window.

V. EXPERIMENTS

A. Batched Scene Generation

We first benchmark the scene generation speed of Sceniris and Scene Synthesizer using a scene (Figure 4(a)) containing three objects: an apple, a banana, and a cabinet. The apple and the cabinet will be randomly placed on the plane, and the banana will be placed inside the upper drawer of the cabinet. The upper and lower drawer joints should be in random states. This scene configuration only contains spatial relationships that are supported in both Sceniris and Scene Synthesizer, so we can perform an apples-to-apples comparison. The benchmark is conducted on a Google Cloud VM with 16 CPU threads (Intel Xeon CPU @ 2.20GHz) and an NVIDIA L4 GPU. Since Scene Synthesizer operates on a single thread, we run Scene Synthesizer [23] on 10 processes simultaneously, and we evenly distribute the scene instances to generate for those processes. Each object is allowed 10 retries when a collision is detected. If a scene instance fails to meet the collision-free condition, the instance will be marked invalid.

Figure 4 shows the system-wise comparison regarding the execution time (b) and the number of valid scenes generated per second (c), between Sceniris and Scene Synthesizer. We vary the total number of scenes to generate from 4 to 16,384. On the execution time, Sceniris can finish the operation on 16,384 scenes in 32s, while Scene Synthesizer can finish 64 scenes in 39s. Considering the number of scenes they operate on, Sceniris achieves about 311 times the efficiency of Scene Synthesizer. This shows Sceniris is a highly efficient system for generating a large batch of random instances for the same configuration. Sceniris achieves another significant efficiency improvement when switching to a warm start case, i.e., only executing the "Generation" step in Figure 2. The warm started Sceniris finishes 16,384 scenes in 2.52s, even shorter than Scene Synthesizer operating on 4 scenes. This shows that we significantly reduced the repeated overhead cost when re-sampling the entire scene. Therefore, Sceniris is not only able to generate a batch of scene instances efficiently, but is also able to continuously generate randomized instances, which is promising in parallel reinforcement learning or data collection scenarios. In Figure 4(c), we show that Sceniris generates about 234 times more valid scenes than Scene Synthesizer in the cold start case and about 2,936 times more in the warm start case.

B. Component Breakdown

We profile the execution time of each key step in the generation procedure in Sceniris and Scene Synthesizer, using the same scene configuration as above. According to Figure 5, Sceniris spends 10.17s on generating **1,024** scenes and Scene Synthesizer takes 2.99s for **1** scene. Sceniris

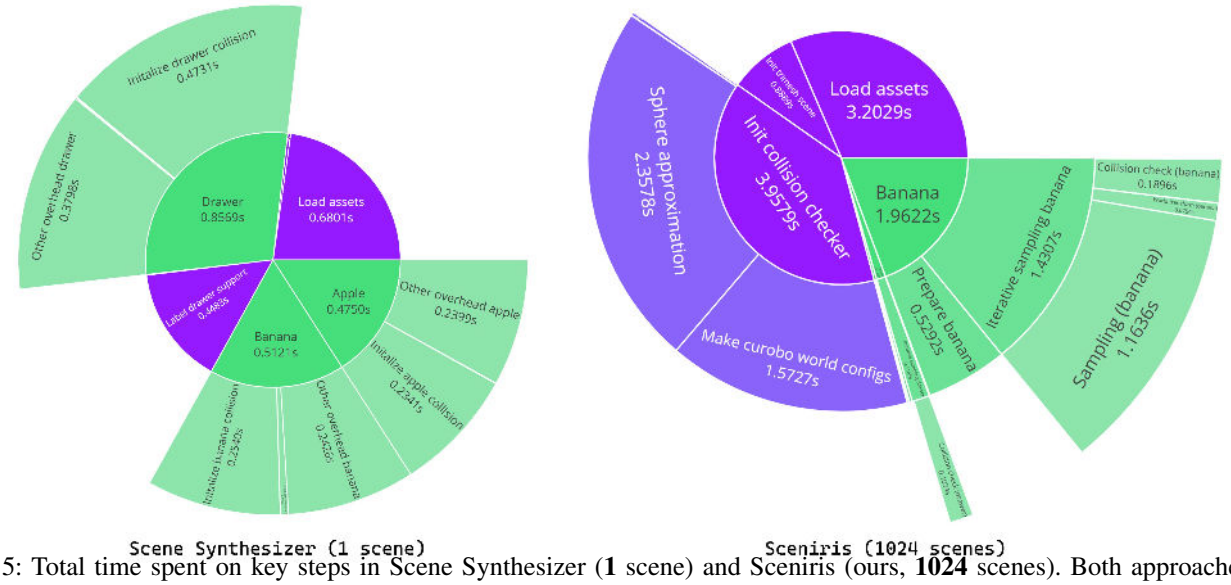

Fig. 5: Total time spent on key steps in Scene Synthesizer (**1** scene) and Sceniris (ours, **1024** scenes). Both approaches use the same scene configuration in Section V-A. Regions in green show the time spent on placing objects.

spends significantly less portion of execution time on placing objects (green) compared to Scene Synthesizer, showing that our successful effort on minimizing the recurrent overhead cost. In comparison, Scene Synthesizer spends a huge portion of time on overhead other than collision checking for objects, shown by "other overhead" on the left of Figure 5. We later find that such overhead in Scene Synthesizer is caused by dumping the object asset to a trimesh scene repeatedly. We then, in Sceniris, implemented a cache mechanism to perform this operation only once. We also perform the benchmark for Sceniris with the reachability check turned on for the apple and the cabinet. We find that the running time is almost the same without the reachability check. Our reachability check only takes a constant ∼0.0001s per query, which is at most 1/20 of the collision check cost.

Sceniris spends a significant amount of time on initializing the collision checker, which we use cuRobo to implement. In the pressure test below, we find that the time for making cuRobo world configurations increases with the number of scenes and some overhead is presented. We believe this overhead will be improved by a better cuRobo interface that improves the creation of world configurations, which is out of scope of this work and will be left for future work.

C. Pressure Test

We conduct a simple pressure test using the same scene configuration in batch scene generation experiment. We find that generating a batch of 524,288 scenes fits in an L4 GPU, taking about 20GB VRAM. It takes 769.67s to finish for the cold start case and has 113,820 invalid scenes, resulting in 533.30 valid scenes/s. This is higher than 408.77 valid scenes/s in Figure 4, showing that our cold start performance is not saturated, and running on a larger batch and a GPU with larger VRAM could continuously improve this performance. For the warm start performance, it achieves 5,279.84 valid scenes/s.

D. Scaling up with Complex Spatial Relationships

Based on the standard benchmark scene configuration above (named Mid), we extend harder scene configurations

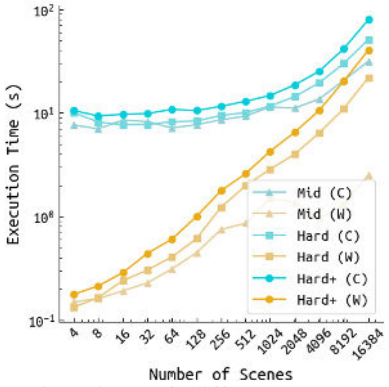

Fig. 6: Execution time of different scene configurations. C: Cold start; W: Warm start. Mid is the basic scene configuration. Hard has 2 complex spatial relationships and Hard+ has 3.

(Hard, Hard+) with our complex spatial relationships and test their generation time. In the Hard scene configuration, we keep the drawer and the banana from mid, and place the apple in front of the drawer at least 0.5 meters away. In addition, the orange should be placed between the drawer and the apple. In Hard+ configuration, we add a mug that should be placed to the right of the drawer within a 0.7m range. The orange should be placed between the drawer, the mug, and the apple. Hard contains 2 complex spatial relationships, and Hard+ contains 3, and the "in between" is harder since there is one additional object involved. In this experiment, we are not able to compare with Scene Synthesizer because it does not support these spatial relationships.

We report the execution speed in Figure 6. We observe that Hard and Hard+ increase about 20-30s execution time over its previous level in the cold start case, and about 20s in the warm start case. This is because some of the complex spatial relationships can not be propagated to the batch by a simple rigid transformation, and Sceniris has to compute the valid sample polygon per scene instance. Nevertheless, Sceniris still achieves about 249 valid scenes/s in Hard+ (cold start), showing that Sceniris has the efficiency advantage compared

to Scene Synthesizer. We believe this could be sped up by batch sampling from heterogeneous polygons on a GPU, which is a potential future work.

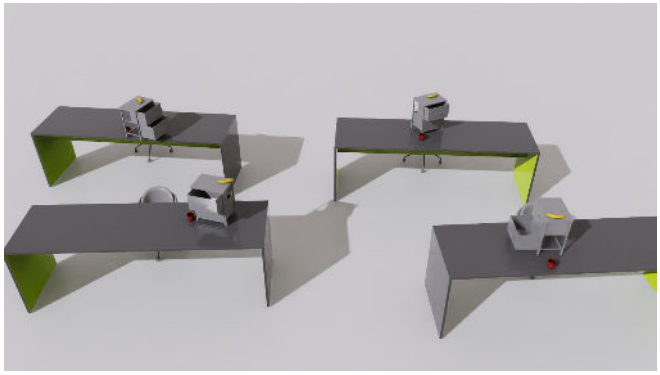

Fig. 7: Example of generated scenes.

E. Qualitative results

In Figure 7, we demonstrate a kitchen scene and a table top scene generated from Sceniris. The scene includes a table, a chair next to the table, a cabinet with two drawers placed on the table, a mug next to the cabinet, and a banana placed on the top of the cabinet. We highlight our capability of maintaining object spatial relationships and randomizing joint states.

VI. CONCLUSIONS

We present Sceniris, a fast procedural scene generation framework using the ideas of parallelization and GPU computing. Sceniris can generate collision-free and robot-reachable scenes that provide rich support for physical AI, learning based scene generation, and 3D understanding applications. Sceniris achieves 200+$\times$ speed-up over the naively multi-processed base approach Scene Synthesizer [23].

REFERENCES

[1] J. Achiam, S. Adler, S. Agarwal, L. Ahmad, I. Akkaya, F. L. Aleman, D. Almeida, J. Altenschmidt, S. Altman, S. Anadkat, *et al.*, "Gpt-4 technical report," *arXiv preprint arXiv:2303.08774*, 2023.

[2] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar, *et al.*, "Llama: Open and efficient foundation language models," *arXiv preprint arXiv:2302.13971*, 2023.

[3] J. Bai, S. Bai, Y. Chu, Z. Cui, K. Dang, X. Deng, Y. Fan, W. Ge, Y. Han, F. Huang, *et al.*, "Qwen technical report," *arXiv preprint arXiv:2309.16609*, 2023.

[4] A. Liu, B. Feng, B. Xue, B. Wang, B. Wu, C. Lu, C. Zhao, C. Deng, C. Zhang, C. Ruan, *et al.*, "Deepseek-v3 technical report," *arXiv preprint arXiv:2412.19437*, 2024.

[5] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby, "An image is worth 16x16 words: Transformers for image recognition at scale," in *International Conference on Learning Representations*, 2021.

[6] M. Oquab, T. Darcet, T. Moutakanni, H. Vo, M. Szafraniec, V. Khalidov, P. Fernandez, D. Haziza, F. Massa, A. El-Nouby, *et al.*, "Dinov2: Learning robust visual features without supervision," *arXiv preprint arXiv:2304.07193*, 2023.

[7] O. Siméoni, H. V. Vo, M. Seitzer, F. Baldassarre, M. Oquab, C. Jose, V. Khalidov, M. Szafraniec, S. Yi, M. Ramamonjisoa, *et al.*, "Dinov3," *arXiv preprint arXiv:2508.10104*, 2025.

[8] J. Shang, K. Schmeckpeper, B. B. May, M. V. Minniti, T. Kelestemur, D. Watkins, and L. Herlant, "Theia: Distilling diverse vision foundation models for robot learning," *arXiv preprint arXiv:2407.20179*, 2024.

[9] M. Ranzinger, G. Heinrich, J. Kautz, and P. Molchanov, "Am-radio: Agglomerative vision foundation model reduce all domains into one," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 12490–12500, 2024.

[10] G. Heinrich, M. Ranzinger, H. Yin, Y. Lu, J. Kautz, A. Tao, B. Catanzaro, and P. Molchanov, "Radiov2.5: Improved baselines for agglomerative vision foundation models," in *Proceedings of the Computer Vision and Pattern Recognition Conference*, pp. 22487–22497, 2025.

[11] J.-B. Alayrac, J. Donahue, P. Luc, A. Miech, I. Barr, Y. Hasson, K. Lenc, A. Mensch, K. Millican, M. Reynolds, *et al.*, "Flamingo: a visual language model for few-shot learning," *Advances in neural information processing systems*, vol. 35, pp. 23716–23736, 2022.

[12] M. Tschannen, A. Gritsenko, X. Wang, M. F. Naeem, I. Alabdulmohsin, N. Parthasarathy, T. Evans, L. Beyer, Y. Xia, B. Mustafa, *et al.*, "Siglip 2: Multilingual vision-language encoders with improved semantic understanding, localization, and dense features," *arXiv preprint arXiv:2502.14786*, 2025.

[13] X. Zhai, B. Mustafa, A. Kolesnikov, and L. Beyer, "Sigmoid loss for language image pre-training," in *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 11975–11986, 2023.

[14] H. Liu, C. Li, Q. Wu, and Y. J. Lee, "Visual instruction tuning," *Advances in neural information processing systems*, vol. 36, pp. 34892–34916, 2023.

[15] A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark, *et al.*, "Learning transferable visual models from natural language supervision," in *International Conference on Machine Learning*, pp. 8748–8763, PMLR, 2021.

[16] K. Grauman, A. Westbury, E. Byrne, Z. Chavis, A. Furnari, R. Girdhar, J. Hamburger, H. Jiang, M. Liu, X. Liu, *et al.*, "Ego4d: Around the world in 3,000 hours of egocentric video," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 18995–19012, 2022.

[17] D. Damen, H. Doughty, G. M. Farinella, S. Fidler, A. Furnari, E. Kazakos, D. Moltisanti, J. Munro, T. Perrett, W. Price, *et al.*, "Scaling egocentric vision: The epic-kitchens dataset," in *Proceedings of the European conference on computer vision (ECCV)*, pp. 720–736, 2018.

[18] 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.

[19] A. O'Neill, A. Rehman, A. Maddukuri, A. Gupta, A. Padalkar, A. Lee, A. Pooley, A. Gupta, A. Mandlekar, A. Jain, *et al.*, "Open x-embodiment: Robotic learning datasets and rt-x models: Open x-embodiment collaboration 0," in *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 6892–6903, IEEE, 2024.

[20] H. Zhang, N. Zantout, P. Kachana, Z. Wu, J. Zhang, and W. Wang, "Vla-3d: A dataset for 3d semantic scene understanding and navigation," *arXiv preprint arXiv:2411.03540*, 2024.

[21] 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.

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

[23] 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.

[24] 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.

[25] 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.

[26] 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.

[27] 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.

[28] 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.

[29] 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.

[30] 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.

[31] N. Pfaff, H. Dai, S. Zakharov, S. Iwase, and R. Tedrake, "Steerable scene generation with post training and inference-time search," 2025.

[32] H. Li, H. Shi, W. Zhang, W. Wu, Y. Liao, L. Wang, L.-h. Lee, and P. Y. Zhou, "Dreamscene: 3d gaussian-based text-to-3d scene generation via formation pattern sampling," in *European Conference on Computer Vision*, pp. 214–230, Springer, 2024.

[33] S. Peng, K. Genova, C. Jiang, A. Tagliasacchi, M. Pollefeys, T. Funkhouser, *et al.*, "Openscene: 3d scene understanding with open vocabularies," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 815–824, 2023.

[34] J. Hou, B. Graham, M. Nießner, and S. Xie, "Exploring data-efficient 3d scene understanding with contrastive scene contexts," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 15587–15597, 2021.

[35] R. Chen, Y. Liu, L. Kong, X. Zhu, Y. Ma, Y. Li, Y. Hou, Y. Qiao, and W. Wang, "Clip2scene: Towards label-efficient 3d scene understanding by clip," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 7020–7030, 2023.

[36] 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)*, pp. 6891–6897, IEEE, 2020.

[37] Z. Chen, G. Wang, and Z. Liu, "Scenedreamer: Unbounded 3d scene generation from 2d image collections," *IEEE transactions on pattern analysis and machine intelligence*, vol. 45, no. 12, pp. 15562–15576, 2023.

[38] J. Shriram, A. Trevithick, L. Liu, and R. Ramamoorthi, "Realmdreamer: Text-driven 3d scene generation with inpainting and depth diffusion," *arXiv preprint arXiv:2404.07199*, 2024.

[39] S. Gillies *et al.*, "Shapely: manipulation and analysis of geometric objects," 2007–.

[40] 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.

[41] E. Todorov, T. Erez, and Y. Tassa, "Mujoco: A physics engine for model-based control," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 5026–5033, 2012.

[42] NVIDIA, "Isaac Sim."

[43] M. Mittal, C. Yu, Q. Yu, J. Liu, N. Rudin, D. Hoeller, J. L. Yuan, R. Singh, Y. Guo, H. Mazhar, A. Mandlekar, B. Babich, G. State, M. Hutter, and A. Garg, "Orbit: A unified simulation framework for interactive robot learning environments," *IEEE Robotics and Automation Letters*, vol. 8, no. 6, pp. 3740–3747, 2023.

[44] E. Coumans and Y. Bai, "Pybullet, a python module for physics simulation for games, robotics and machine learning." `http://pybullet.org`, 2016–2021.

[45] R. McLean, E. Chatzaroulas, L. McCutcheon, F. Röder, T. Yu, Z. He, K. R. Zentner, R. Julian, J. K. Terry, I. Woungang, N. Farsad, and P. S. Castro, "Meta-world+: An improved, standardized, rl benchmark," 2025.

[46] Y. Zhu, J. Wong, A. Mandlekar, R. Martín-Martín, A. Joshi, S. Nasiriany, Y. Zhu, and K. Lin, "robosuite: A modular simulation framework and benchmark for robot learning," in *arXiv preprint arXiv:2009.12293*, 2020.

[47] E. Rohmer, S. P. N. Singh, and M. Freese, "Coppeliasim (formerly v-rep): a versatile and scalable robot simulation framework," in *Proc. of The International Conference on Intelligent Robots and Systems (IROS)*, 2013. www.coppeliarobotics.com.

[48] H. Geng, F. Wang, S. Wei, Y. Li, B. Wang, B. An, C. T. Cheng, H. Lou, P. Li, Y.-J. Wang, Y. Liang, D. Goetting, C. Xu, H. Chen, Y. Qian, Y. Geng, J. Mao, W. Wan, M. Zhang, J. Lyu, S. Zhao, J. Zhang, J. Zhang, C. Zhao, H. Lu, Y. Ding, R. Gong, Y. Wang, Y. Kuang, R. Wu, B. Jia, C. Sferrazza, H. Dong, S. Huang, Y. Wang, J. Malik, and P. Abbeel, "Roboverse: Towards a unified platform, dataset and benchmark for scalable and generalizable robot learning," 2025.

[49] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, "CARLA: An open urban driving simulator," in *Proceedings of the 1st Annual Conference on Robot Learning*, pp. 1–16, 2017.

[50] X. Puig, E. Undersander, A. Szot, M. D. Cote, T.-Y. Yang, R. Partsey, R. Desai, A. W. Clegg, M. Hlavac, S. Y. Min, V. Vondruš, T. Gervet, V.-P. Berges, J. M. Turner, O. Maksymets, Z. Kira, M. Kalakrishnan, J. Malik, D. S. Chaplot, U. Jain, D. Batra, A. Rai, and R. Mottaghi, "Habitat 3.0: A co-habitat for humans, avatars and robots," 2023.

[51] A. Szot, A. Clegg, E. Undersander, E. Wijmans, Y. Zhao, J. Turner, N. Maestre, M. Mukadam, D. Chaplot, O. Maksymets, A. Gokaslan, V. Vondrus, S. Dharur, F. Meier, W. Galuba, A. Chang, Z. Kira, V. Koltun, J. Malik, M. Savva, and D. Batra, "Habitat 2.0: Training home assistants to rearrange their habitat," 2022.

[52] M. Savva, A. Kadian, O. Maksymets, Y. Zhao, E. Wijmans, B. Jain, J. Straub, J. Liu, V. Koltun, J. Malik, D. Parikh, and D. Batra, "Habitat: A platform for embodied ai research," in *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 9338–9346, 2019.

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

[54] 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.

[55] O. Mees, L. Hermann, E. Rosete-Beas, and W. Burgard, "Calvin: A benchmark for language-conditioned policy learning for long-horizon robot manipulation tasks," *IEEE Robotics and Automation Letters (RA-L)*, vol. 7, no. 3, pp. 7327–7334, 2022.

[56] 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.

[57] X. Li, K. Hsu, J. Gu, K. Pertsch, O. Mees, H. R. Walke, C. Fu, I. Lunawat, I. Sieh, S. Kirmani, S. Levine, J. Wu, C. Finn, H. Su, Q. Vuong, and T. Xiao, "Evaluating real-world robot manipulation policies in simulation," *arXiv preprint arXiv:2405.05941*, 2024.

[58] 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.

[59] Y. Mu, T. Chen, S. Peng, Z. Chen, Z. Gao, Y. Zou, L. Lin, Z. Xie, and P. Luo, "Robotwin: Dual-arm robot benchmark with generative digital twins (early version)," *arXiv preprint arXiv:2409.02920*, 2024.

[60] G. Lumer-Klabbers, J. O. Hausted, J. L. Kvistgaard, H. D. Macedo, M. Frasheri, and P. G. Larsen, "Towards a digital twin framework for autonomous robots," in *2021 IEEE 45th Annual Computers, Software, and Applications Conference (COMPSAC)*, pp. 1254–1259, 2021.

[61] T. Dai, J. Wong, Y. Jiang, C. Wang, C. Gokmen, R. Zhang, J. Wu, and L. Fei-Fei, "Automated creation of digital cousins for robust policy learning," in *Conference on Robot Learning (CoRL)*, 2024.

[62] B. M. Öcal, M. Tatarchenko, S. Karaoğlu, and T. Gevers, "Sceneteller: Language-to-3d scene generation," in *European Conference on Computer Vision*, pp. 362–378, Springer, 2024.

[63] S. Zhang, Y. Zhang, Q. Zheng, R. Ma, W. Hua, H. Bao, W. Xu, and C. Zou, "3d-scenedreamer: Text-driven 3d-consistent scene generation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 10170–10180, 2024.

[64] J. Zhang, X. Li, Z. Wan, C. Wang, and J. Liao, "Text2nerf: Text-driven 3d scene generation with neural radiance fields," *IEEE Transactions on Visualization and Computer Graphics*, vol. 30, no. 12, pp. 7749–7762, 2024.

[65] T.-Y. Lin, C.-H. Lin, Y. Cui, Y. Ge, S. Nah, A. Mallya, Z. Hao, Y. Ding, H. Mao, Z. Li, *et al.*, "Genusd: 3d scene generation made easy," in *ACM SIGGRAPH 2024 Real-Time Live!*, pp. 1–2, 2024.

[66] A. Chang, W. Monroe, M. Savva, C. Potts, and C. D. Manning, "Text to 3d scene generation with rich lexical grounding," *arXiv preprint arXiv:1505.06289*, 2015.

[67] A. Raistrick, L. Lipson, Z. Ma, L. Mei, M. Wang, Y. Zuo, K. Kayan, H. Wen, B. Han, Y. Wang, *et al.*, "Infinite photorealistic worlds using procedural generation," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 12630–12641, 2023.

[68] 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*, pp. 21783–21794, 2024.

[69] A. Joshi, B. Han, J. Nugent, Y. Zuo, J. Liu, H. Wen, S. Alexandropoulos, T. Sun, A. Raistrick, G. Liu, *et al.*, "Infinigen-sim: Procedural generation of articulated simulation assets," *arXiv preprint arXiv:2505.10755*, 2025.

[70] "Unreal engine procedural content generation."

[71] 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.