DOI: 10.1111/cgf.14657
Pacific Graphics 2022
N. Umetani, E. Vouga, and C. Wojtan
(Guest Editors)

Volume 41 (2022), Number 7

MINERVAS: Massive INterior EnviRonments VirtuAl Synthesis

Haocheng Ren[1†], Hao Zhang[1†], Jia Zheng[2], Jiaxiang Zheng[2], Rui Tang[2‡], Yuchi Huo[1], Hujun Bao[1] and Rui Wang[1‡]

[1]State Key Lab of CAD&CG, Zhejiang University, [2]Manycore Tech Inc.

Abstract

With the rapid development of data-driven techniques, data has played an essential role in various computer vision tasks. Many realistic and synthetic datasets have been proposed to address different problems. However, there are lots of unresolved challenges: (1) the creation of dataset is usually a tedious process with manual annotations, (2) most datasets are only designed for a single specific task, (3) the modification or randomization of the 3D scene is difficult, and (4) the release of commercial 3D data may encounter copyright issue. This paper presents MINERVAS, a Massive INterior EnviRonments VirtuAl Synthesis system, to facilitate the 3D scene modification and the 2D image synthesis for various vision tasks. In particular, we design a programmable pipeline with Domain-Specific Language, allowing users to select scenes from the commercial indoor scene database, synthesize scenes for different tasks with customized rules, and render various types of imagery data, such as color images, geometric structures, semantic labels. Our system eases the difficulty of customizing massive scenes for different tasks and relieves users from manipulating fine-grained scene configurations by providing user-controllable randomness using multi-level samplers. Most importantly, it empowers users to access commercial scene databases with millions of indoor scenes and protects the copyright of core data assets, e.g., 3D CAD models. We demonstrate the validity and flexibility of our system by using our synthesized data to improve the performance on different kinds of computer vision tasks. The project page is at https://coohom.github.io/MINERVAS.

CCS Concepts

• ***Computing methodologies*** → *Graphics systems and interfaces;*

1. Introduction

Recent advances in various computer vision tasks have shown the tremendous capabilities of deep learning algorithms. Meanwhile, datasets have demonstrated their importance in affecting the performance of these data-driven algorithms. Most datasets [SHKF12, CSC*20, SLX15, DCS*17, CDF*17, XZH*18, XOT13] are built from the real world, and contain imagery or 3D data (e.g., scanned point clouds or meshes). However, collecting the data and manually annotating is a tedious and time-consuming process.

The creation of synthetic datasets with a lower cost has become an essential endeavor for accomplishing learning tasks to address these issues. Many synthetic datasets have been presented already. Some datasets release images or videos [MHLD17, LS18, LSM*18, ZZL*20] for specific vision tasks, others release 3D scenes [HPSC16, LYS*21, SYZ*17, FCG*21, RP21] to give users more flexibility to synthesize images or videos. However, given the complexity of 3D scene editing and rendering, modifying 3D scenes to augment datasets for training different vision tasks is a complicated process requiring relevant expertise. Another considerable obstacle to the synthetic dataset is the lack of high-quality 3D assets. Although some commercial companies have massive 3D scene data, they cannot provide it directly to researchers due to copyright restrictions [SYZ*17] and the huge data storage size (e.g., more than hundreds of Terabytes of 3D CAD models). Hence, a new requirement arises that allowing users to access these massive 3D scenes while protecting the core assets of commercial companies.

In this paper, we introduce MINERVAS, a Massive INterior EnviRonments VirtuAl Synthesis system, to facilitate various vision problems by providing an imagery data synthesis platform. Our system leverages the 3D scenes from the commercial indoor scene database and generates large-scale task-specific synthetic data. To help users customize data, we design a programmable pipeline with Domain-Specific Language (DSL) to control the data synthesis process, such as filtering and editing 3D scenes, defining different desired outputs. Moreover, since the diversity of the data is crucial for learning-based methods, we provide user-controllable samplers at different levels (i.e., scene, entity, and image) to support domain randomization within our system. The proposed system makes the data synthesis for different vision tasks simple and flexible. In ad-

[†] Equal contributions.

[‡] Corresponding authors: Rui Tang, ati@qunhemail.com; Rui Wang, rwang@cad.zju.edu.cn.

dition, the core 3D data assets are not released directly, thereby avoiding potential copyright issues [SYZ*17].

We showcase the validity and flexibility of our system with different vision tasks, including room layout estimation, semantic segmentation, and depth estimation. Our system can easily augment the training data for these tasks, where it only takes 12 hours to generate 1 million photorealistic images. The experimental results show that performances improve on these tasks.

In summary, the main contributions of our work are: (1) We show the possibility to contribute the power of the high-quality commercial scene database to the community while protecting the copyright of the core assets (e.g., 3D CAD models) for the company. (2) We present a multi-stage programmable pipeline with DSL to generate synthetic images in batch processing. (3) We develop multi-level samplers to improve diversity at scene, entity, and image levels. (4) We show the potential of our system by demonstrating the usefulness of our synthetic data on several vision tasks.

2. Related Work

Interior datasets. Recently, many interior datasets [ASZ*16, DCS*17, CDF*17, XZH*18, SWM*19, FRS*12, HPSC16, SYZ*17, ADD*19, WZW*20, LYS*21, FJG*21, FCG*21, RP21] have been proposed to facilitate research on various indoor tasks. Table 1 shows existing datasets, which can be broadly divided into two distinct categories: real-world datasets and synthetic datasets.

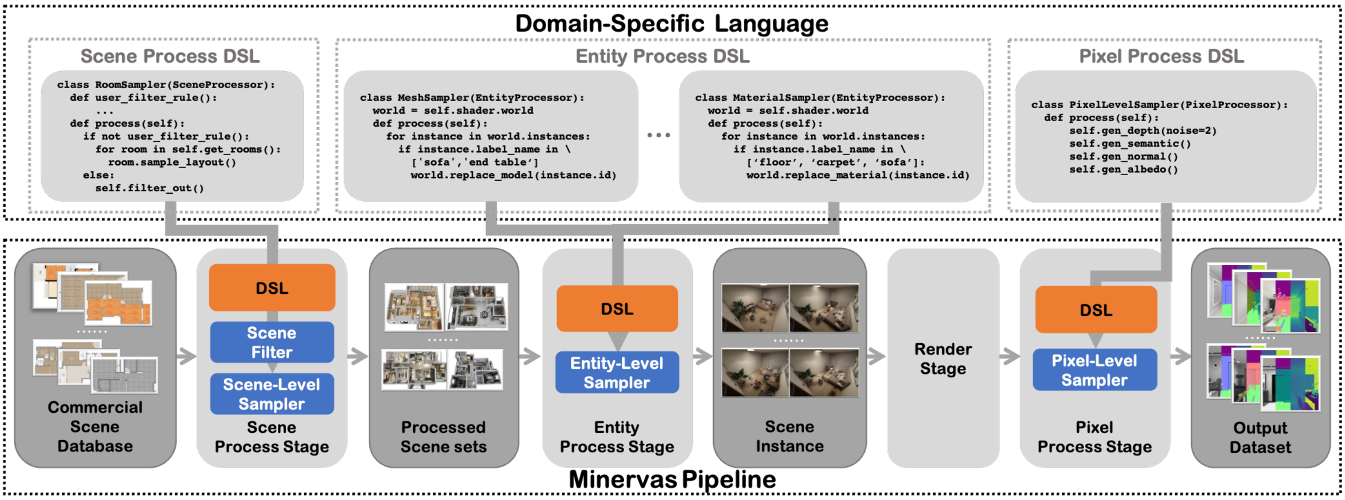
Real-world datasets are usually collected in the real world using RGB-D sensors (such as Microsoft Kinect) and represented as RGB-D sequences [SHKF12, SEE*12, JKJ*11, SLX15], point clouds [KAJS11, XOT13, LBF14, ASZ*16, XZH*18] or meshes [HPN*16, SCH*16, CDF*17, ASZS17, DCS*17, SWM*19]. NYUv2 [SHKF12] collected 464 RGB-D sequences and per-pixel annotations in selected 1449 frames. SUN 3D [XOT13] recovered the full 3D extent of 254 spaces as point clouds instead of a limited set of 2D views. SceneNN [HPN*16] reconstructed triangles meshes from more than 100 scenes with fine-grained annotation. Several works utilize the Matterport system to reconstruct larger scenes as dense meshes [CDF*17, ASZS17] or point clouds [ASZ*16, XZH*18]. However, those datasets provide a limited number of scenes due to the complex capture process. ScanNet [DCS*17] takes one step further by reconstructing 1513 scenes, which is significantly larger than previous works. It utilizes a portable iPad-based RGB-D camera to capture scenes, which helps to involve more untrained users. However, the mesh quality and the accuracy of semantic segmentation are not high enough. On the contrary, Replica [SWM*19] focused on the quality instead of the dataset size. They use a custom-built RGB-D capture rig to obtain 18 high-quality reconstructed scenes. Dense meshes, high-resolution HDR textures, and semantic segmentation are also included. Most recently, HM3D [RGW*21] provides 1000 high-quality reconstructed scenes, but the semantic annotation is not included. Due to the complexity of data collection and annotation, the sizes of the real-world datasets are usually tiny compared to synthetic datasets, as shown in Table 1. The dataset size constrains its diversity. Furthermore, the real-world dataset is usually not editable for augmentation. Besides, the captured or reconstructed data from natural environments usually consists of inherent errors.

Table 1: *A comparison of existing indoor scene datasets. Notations: 3D (including point cloud or meshes), O (object detection), U (scene understanding), S (image synthesis), M (structured 3D modeling).*

Real Dataset	Format	#Scenes	Editable	#Category	Application
Cornell RGB-D [KAJS11]	Image+3D	52	✗	—	U O
NYUv2 [SHKF12]	Image	464	✗	894	U O
TUM [SEE*12]	Image	47	✗	—	U O
SUN 3D [XOT13]	Image+3D	254	✗	—	U O
B3DO [JKJ*11]	Image	75	✗	50	U O
RGBDv2 [LBF14]	3D	17	✗	51	U O
SUN RGB-D [SLX15]	Image	—	✗	800	U O
SceneNN [HPN*16]	Image+3D	100	✗	40	U O
PiGraphs [SCH*16]	Image+3D	26	✗	—	U O
S3DIS [ASZ*16]	3D	265	✗	—	U O
Stanford 2D-3D-S [ASZS17]	Image+3D	279	✗	13	U O
Matterport3D [CDF*17]	Image+3D	90	✗	40	U O
ScanNet [DCS*17]	Image+3D	1513	✗	≈1000	U O
Gibson [XZH*18]	Image+3D	572	✗	80	U O
Replica [SWM*19]	3D	18	✗	88	U O
360-Indoor [CSC*20]	Image	—	✗	37	U O
HM3D [RGW*21]	Image+3D	1000	✗	—	U O
Synthetic Dataset	Format	#Scenes	Editable	#Category	Application
Stanford Scenes [FRS*12]	3D	130	✓	6	U O S
SUNCG [SYZ*17]	Image+3D	45622	✓	84	U O S
SceneNet [HPSC16]	3D	57	✓	218	U O S
AI2-THOR [KMH*17]	3D	120	✓	102	U O
SceneNet-RGB-D [MHLD17]	Image	57	✓	255	U O
CGIntrinsics [LS18]	Image	20K+	✗	—	U O
InteriorNet [LSM*18]	Image	22M	✗	158	U O S
House3D [WWGT18]	Image	45K	✗	84	U O
RobotriX [GGMGO*18]	Image+3D	16	✓	38	U O S
DeepFurniture [LZZ*19]	Image	—	✗	11	U O
Scan2CAD [ADD*19]	3D	1506	✓	37	U O S
Structured3D [ZZL*20]	Image+Structure	3500	✗	40	U O S M
TartanAir [WZW*20]	Image+3D	30	✓	—	U O S
OpenRooms [LYS*21]	Image+3D	1506	✓	24	U O S
3D-FUTURE [FJG*21]	Image+3D	5000	✓	34	U O S
3D-FRONT [FCG*21]	Image+3D	6813	✓	—	U O S
Hypersim [RP21]	Image+3D	461	✓	—	U O S
MINERVAS (Ours)	Image+Structure	**50M**	✓	285	U O S M ...

With the advantages of large-scale, easy-annotation, and editable, many synthetic datasets are introduced to improve performance on several tasks [ZSY*17, LS18, ZZL*20]. SUNCG [SYZ*17] was a widely used artist-designed synthetic dataset. However, SUNCG is unavailable online due to copyright problems. SceneNet [HPSC16] and Scan2CAD [ADD*19] have the capability of modification by sampling objects. However, these datasets are lack realism and semantic diversity. Recent synthetic datasets only release a few 3D data [FRS*12, HPSC16, KMH*17, GGMGO*18, WZW*20, RP21] or imagery data [LSM*18, MHLD17, LS18, WWGT18, LZZ*19]. Most recently, OpenRooms [LYS*21] release an interior synthetic dataset built upon publicly available real scanned datasets, but the diversity of scenes and objects are not as good as artist-created datasets such as SUNCG. We build upon the professional artist-created interior designs, like InteriorNet [LSM*18] and Structured3D [ZZL*20], but our system allows users to manipulate the scenes and synthesize task-specific data using DSL. Furthermore, our database has the largest number of scenes among existing datasets as shown in Table 1.

Though the works discussed above contribute many interior datasets, few works focus on the copyright protection of 3D assets. A classical and useful way to protect 3D assets is remote render-



```
class RoomSampler(SceneProcessor):
  def user_filter_rule():
      ...
  def process(self):
    if not user_filter_rule():
      for room in self.get_rooms():
        room.sample_layout()
    else:
      self.filter_out()
```

```
class MeshSampler(EntityProcessor):
  world = self.shader.world
  def process(self):
    for instance in world.instances:
      if instance.label_name in \
        ['sofa','end table']
        world.replace_model(instance.id)
```

```
class MaterialSampler(EntityProcessor):
  world = self.shader.world
  def process(self):
    for instance in world.instances:
      if instance.label_name in \
        ['floor', 'carpet', 'sofa']:
        world.replace_material(instance.id)
```

```
class PixelLevelSampler(PixelProcessor):
  def process(self):
      self.gen_depth(noise=2)
      self.gen_semantic()
      self.gen_normal()
      self.gen_albedo()
```

Figure 1: *Overview of MINERVAS system. Our system has four stages to synthesis scenes and render images, DSLs are injected into stages for customized user control.*

ing [KTL*04, KL05]. Several strategies like adding perturbations are also proposed [KL05] to prevent the accurate reconstruction of 3D models. Recently, MeshChain [PHY21] utilizes blockchain technology to protect 3D models and enable data source authentication. Our system can protect 3D assets via remote rendering.

Automatic scene generation. Applications in computer graphics and computer vision have inspired many works on automatic scene editing, especially 3D scene generation [MSL*11, YYT15, KK18, RWL19, WLW*19, LPX*19, ZYM*20, ZZX*21, ZXZ21]. These methods assist users by suggesting novel furniture layouts under given constraints. Kán and Kaufman [KK18] proposed a new automatic furniture arrangement method using greedy cost optimization and achieved good interactive speed. Recently, several works have shown that probabilistic programming language can be used to define distribution to generate scenes, such as WebPPL [GS14], Picture [KKTM15], and Scenic [FDG*19]. More recently, Jiang et al. [JQZ*18] proposed a configurable approach to generate synthetic scenes, which is in some sense the most closely related to our work. Nevertheless, their work focuses on scene synthesis rather than a system that enables batch processing and user-controllable 3D scene editing from a large-scale database.

Programmable image synthesis. Several works [JHHB16, KWR*16, RVRK16] have involved adapting the real-time rendering techniques used in computer games to generate the image and video datasets. Several simulators with open-source API [DRC*17, BDK*18,XZH*18,SKM*19] have been proposed for various tasks. Habitat-Sim [SKM*19] provided an environment for training embodied AI agents, especially for navigation tasks. However, our system aims to generate imagery datasets for different vision tasks. Furthermore, it is mainly built on real-world datasets (e.g., Replica [SWM*19], Matterport3D [CDF*17], Gibson [XZH*18]), which makes it hard to edit scenes like synthetic data, as we discussed above. These methods efficiently simulate a single scene, but they are not designed for easy scene modification and data generation in batch processing.

Most recently, several systems [DSW*19, MTL*21, GSA*21, ESMZ21, BCD*21, GBB*22] are proposed to generate synthetic dataset from collections of 3D assets. Though the overall goal is similar, our system differs from them in several aspects. BlenderProc [DSW*19] provides a configurable system that can generate photorealistic large-scale imagery datasets from public 3D scene databases, but it has limited scriptable interfaces. NviSII [MTL*21], ThreeDWorld [GSA*21] and Kubric [GBB*22] uses python API to address this problem. It enables fine-grained control of the scene, but it brings a tedious coding process lacking some high-level built-in functions for randomization. Our system provides a user-friendly DSL, considering both flexibilities with the Python programming language and ease of use with built-in randomization support. Our multi-stage pipeline design also allows users to modify the parts they are interested in and prevent writing the complete data generation code.

Aside from the system design, 3D assets in our system also differ from them. Omnidata [ESMZ21] generates imagery datasets using 3D scan data from the real world. Since it uses the 3D scans database, scenes are not editable thus lack further domain randomization (e.g., novel furniture arrangement) like our artist-created synthetic scenes. Unity Perception [BCD*21] also provides a flexible interface for users. However, it uses the Unity Asset Store as the database, the quality of assets is not ensured, and manually annotating assets is still required. The concurrent work Kubric [GBB*22] mostly relies on existing public 3D assets. Instead, there is a high-quality large-scale commercial scene database behind our system for users to access as an online service. Moreover, some works [BCD*21,GSA*21] render dataset based on rasterization for fast rendering, while others [DSW*19, MTL*21, GBB*22] rely on a path tracer for photorealistic rendering. Our system provides both types of renderers for users to trade-off efficiency and realism.

3. System Overview

MINERVAS system aims to provide a platform for users to easily and efficiently synthesize the large-scale, high-quality interior imagery dataset. To this end, we introduce a novel programmable dataset synthesis pipeline, as shown in Figure 1.

3.1. Design Goals

Our system was guided by the following goals:

- **High-quality data:** The quality of data is important to deep learning. The higher level of realism could reduce the domain gap [ZSY*17] between the synthetic dataset and the real world. The creation of a high-quality synthetic imagery dataset requires both elaborately designed 3D assets and state-of-the-art photorealistic rendering techniques.
- **Large-scale data:** More data usually brings better results. One important goal of our system is to provide the largest number of 3D interior scenes among existing synthetic interior datasets.
- **Diverse data:** Domain randomization is a common technique to boost the model performance in the real case [TFR*17]. To increase the diversity of the synthesized data, we need to provide both the diverse 3D scenes in the database and the ability to easily augment these 3D scenes in our system.
- **Full user control:** As our system aims to facilitate a broad spectrum of vision tasks, users need to control the system at finer granularity to meet their requirements.
- **Ease of use:** With the guarantee of full user control, we also expect our system to provide a user-friendly control interface, reducing the time users are familiar with the system.
- **Fast synthesis:** Rendering a large-scale synthetic dataset is a tedious and time-consuming process [ZSY*17]. We expect our system can generate synthetic datasets as fast as possible while ensuring the quality of the data.

3.2. Pipeline Overview

Our system pipeline consists of four stages, as shown in Figure 1. In the Scene Process Stage, users first select a set of scenes from the commercial scene database and rearrange the furniture layout for scene-level randomization. Then, in the Entity Process Stage, users could manipulate a single object for entity-level randomization. Next, 2D imagery data is rendered in the Render Stage. Finally, in the Pixel Process Stage, the users could apply pixel-wise modifications to the 2D renderings, such as simulating sensor noises.

To improve the diversity of the synthesized dataset, we introduce a randomization mechanism in our system called the sampler. We design multi-level samplers for different stages according to the types of data being processed. Users could control the data synthesis process in all stages of the pipeline via DSL at different levels of granularity control while maintaining ease of use.

3.3. Key Design Decisions

To meet the goals in Section 3.1, the following features are considered when designing our system.

Large-scale high-quality commercial scene database. The quality of 3D assets largely affects the quality of the output data. To achieve the best quality and largest number of 3D scenes as we can, we leverage the online commercial 3D scene database from the interior design company . The underlying database consists of 50 million house designs created by professional interior designers. The dataset has diverse room layouts, furniture arrangements, and lighting setups. Figure 2 shows the statistics of the database. The house layout encompasses a wide range of realistic residential spaces. The database also includes 1 million Computer-Aided Design (CAD) furniture models. These models have been categorized into about 300 categories. We also provide category mappings to commonly-used label sets, such as NYUv2 40 label set [SHKF12]. Each model has physically-based spatially-varying material. This commercial 3D database empowers the ability of our system to generate diverse data by sampling furniture, material, and lighting.

Modular system architecture. By examining the entire process of generating one synthetic image, we found it mainly involves three types of data, i.e., 3D scenes in the database, objects in scenes, and rendered images. For each type of data, processing operations should be different and specific computations need to be customized for different vision tasks. Therefore, we divided the pipeline into four stages. Each stage processes a single type of data in batch. Modulation design improves the flexibility and throughput of our system.

Diverse synthesis results. The diversity and variety of data for training are important for data-driven algorithms. Large-scale datasets with diverse samples are essential to ensure that the algorithm performs adequately in unusual circumstances or challenging cases. In some cases, the users may also need to augment data by varying specific parameters in a scene. Our system enables user-controllable randomness through samplers at different granularity levels of the scenes, entities, and images to improve the data diversity (Section 4).

User customization. Different tasks have different requirements on synthetic datasets. Flexible user customization is an essential feature of our system to meet the needs of different vision tasks. Specifically, we employ a DSL to plug in custom computation in different stages, thus letting users control the process of image generation for their tasks. Additionally, users can utilize the DSL to modify the built-in samplers, letting the randomization be customized (Section 5). In addition to customization using DSL, users who do not need complex filtering or randomization procedure can use the GUI of our system to customize their generation process while reducing the time of learning the DSL. Compared with DSL's full user customization, the GUI scarifies some freedom but makes our system easier to use.

Cloud-based rendering. As the data synthesis is a compute-intensive task, we deployed our system on a cloud-based cluster with more than 5000 computation nodes, each one composed of the Intel Xeon CPU with 64 cores. Our system takes about 12 hours to

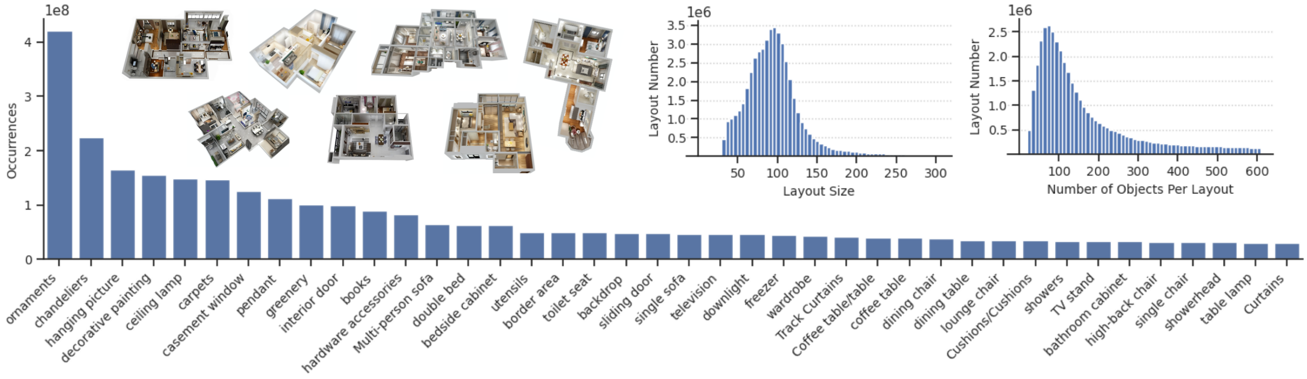


Figure 2: *Statistics of the scene database. We show the 40 most frequent categories in the database, the distribution of room size (m^2) per layout, the distribution of the number of objects per layout.*

generate a large-scale imagery dataset with 1 million views based on the powerful cluster. In some cases, users want to balance rendering costs and time. For example, some users do not need fast computation on the powerful cluster. We also provide the option to render on the local machine to facilitate this.

Online service. Our new programmable system pipeline allows us to implement it as an online service, which can further reduce the tedious process of deploying the system on local machines. Note that implementing our system as an online service can also protect the copyright of these commercial assets [KL05], because the users cannot directly download the core assets (e.g., 3D CAD models) of the database via our online service. The researchers could redistribute the data generated on our platform, and we promise the generated data is free from copyright issues. As far as we know, it is the first time to introduce such a massive scale commercial interior 3D scene database to the research community without copyright issues.

4. Random Scene Synthesis

One advantage of synthetic datasets over real-world datasets is that users can freely edit each part of the scenes to meet their dataset generation needs and enable domain randomization. However, editing scenes usually requires relevant expertise, and it is usually tedious to edit massive numbers of scenes. Also, for large-scale dataset synthesis, it is impractical to configure the parameters for each scene manually.

In our system, we introduce a randomness mechanism called sampler to ease the difficulty of editing the scene and improve the diversity of the synthesized datasets. According to the types of data being processed in the different stages of the pipeline (Figure 1), we design multi-level samplers to automatically perform scene-level furniture arrangement, entity-level attributes modification, and pixel-level processing.

4.1. Scene Process Stage

In the Scene Process Stage, users first select scenes from the commercial scene database according to user-provided conditions, e.g., area, the number of rooms, and room types. To support the mega-scale scene database, we use MongoDB as the underlying database program. We store all structured metadata of the scene in the database, such as scene graph and scene attributes (e.g., room size, the amount of furniture in the scene).

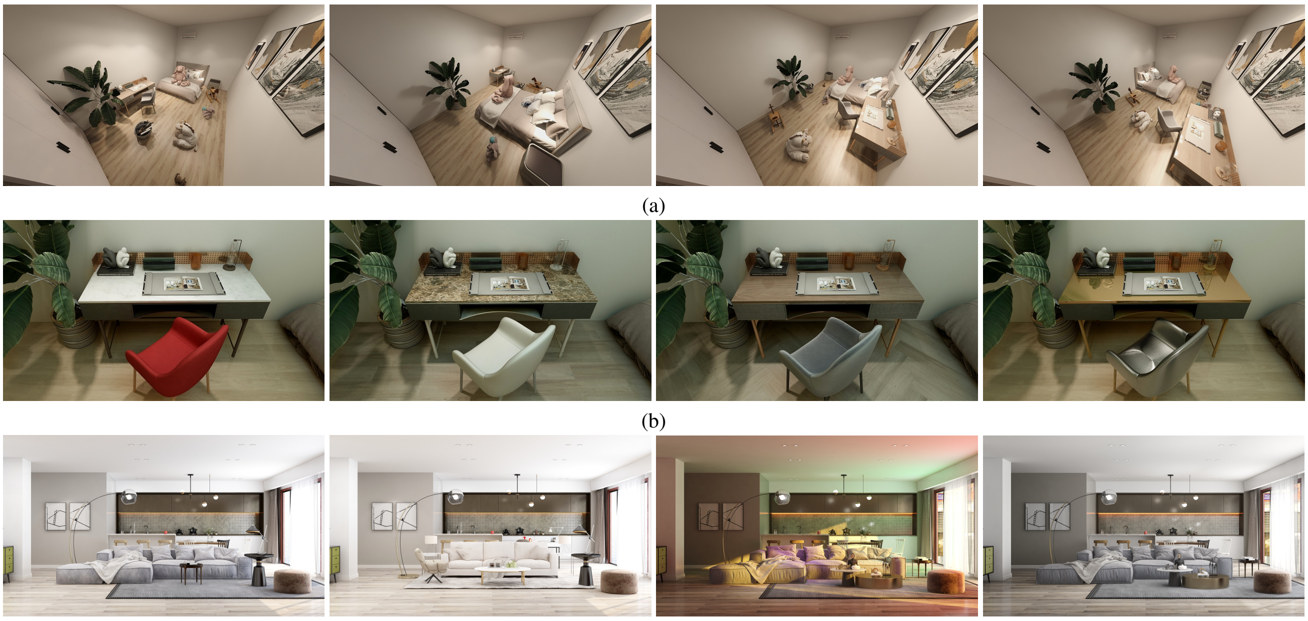
Given a selected scene, we provide a scene-level sampler to generate novel furniture arrangements. Specifically, we adopt the approach [KK18] by taking into account existing furniture and cost functions on the clearance, circulation, group relationships, alignment, distribution, and rhythm. We iteratively generate new arrangements by randomly moving each furniture instance, i.e., changing its positions and orientations. After each iteration, a cost value is calculated to accept or reject the generated layout. The number of iterations is usually determined by the amount of furniture in the scene. Users could easily generate various reasonable furniture arrangements with this scene-level sampler. Figure 3 (a) shows several layouts generated by the scene-level sampler.

The processed scenes are then sent to the Entity Process Stage for further processing.

4.2. Entity Process Stage

The Entity Process Stage is designed for processing each object in the scene.

We provide some entity-level samplers to randomize attributes of each entity, including furniture (e.g., CAD model, material, transformation), light (e.g., intensity, color), and camera (e.g., camera model, transformation). Depending on the characteristics of each attribute, we use different types of distributions for different attributes. For example, we utilize uniform, Gaussian distributions to describe continuous attributes (e.g., position, light), discrete distribution for discrete attributes (e.g., material), and learning-based distributions for other attributes (e.g., CAD model). Figure 3 (b-d)


(a)

(b)

(c) (d)

Figure 3: *Results generated by different samplers. (a) is the results of the scene-level sampler. (b–d) are the results of the entity-level samplers. (b) is the results of randomizing materials. (c) is the results of replacing meshes. (d) is the results under different lighting.*

shows the results of sampling different attributes using the entity-level sampler.

Here, the common attributes are shown as follows:

- **Camera.** We support various camera models (e.g., orthographic, perspective, and panoramic camera models) and camera parameters (e.g., field of view, image resolution).
- **Trajectory.** We explicitly define trajectory as an attribute of the movable entity, e.g., camera. To randomly generate roaming trajectories in 3D space, we implement a two-physical-bodies trajectory simulation method [HPSC16], i.e., taking a uniform sampler to generate random forces, creating a series of random pivots of the trajectory. Users could also generate the handcrafted trajectory by specifying key points.
- **Light.** Users could control the intensity and color temperature of each light. We assume these attributes follow the uniform distribution. We also support day and night lighting modes, which restrict the range of intensity and color to produce natural lighting.
- **Material**. Given a material category, the sampler uniformly samples materials of that category from a built-in table.
- **CAD model.** In practice, uniform sampling CAD models may not produce good results. Instead, we use an internal similarity map among all CAD models to guide the sampler. More specifically, based on the preview image of CAD model, we use VGGNet [SZ15] to construct a feature vector for each CAD model and perform similarity retrieval based on feature vectors according to their Euclidean metric. When replacing the CAD model, we place a new CAD model in the original place and make sure the object touches the floor or a supported object.
- **Transformation.** The sampler could sample rotation and translation within a given range from a normal or a uniform distribution. To avoid intersections, users can enable collision detection using rejection sampling in the sampler.

Furthermore, this stage supports to export the non-imagery data (e.g., 3D structures [ZZL*20]) to fulfill specific task requirements such as room layout estimation. The non-imagery data could include any meta information except the raw data of 3D mesh and material.

The generated scenes are then sent to the Render Stage.

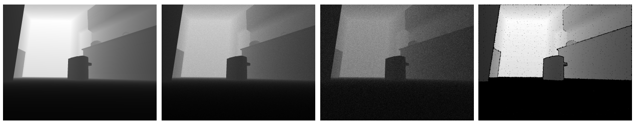
4.3. Render Stage

In Render Stage, the system uses the generated scenes to generate 2D renderings. Our system provides both rasterization-based and path-tracing-based renderers to balance efficiency and realism. This stage is also configurable, e.g., the number of samples, the number of light bounces. The renderer supports different kinds of imagery data, e.g., color, depth, normal, semantic, instance. The rendered pixel-wise ground truths could facilitate various indoor scene understanding tasks. After this stage, the 2D renderings will be sent to the Pixel Process Stage.

4.4. Pixel Process Stage

The Pixel Process Stage allows users to apply pixel-wise processing operations on the imagery data.

We provide several built-in functions, such as mapping the semantic labels to the desired label set, simulating different types of depth noise (e.g., Gaussian noise, Poisson noise, salt-and-pepper



(a) (b) (c) (d)

Figure 4: *Results generated by the pixel-level sampler. (a) is the ground-truth depth output, whilst (b) add Poisson noise, (c) add Gaussian noise, and (d) add Gaussian shifts (Kinect noise).*

noise and Kinect noise [BM13, HWMD14, BRHS14] as shown in Figure 4). Users could also apply arbitrary customized image processing methods to the 2D renderings.

5. DSL in MINERVAS

We design the DSL under two principles: flexibility and ease of use. For flexibility, we use the Python programming language as the host to build our internal DSL. Using its mechanism for assignment, arithmetic, loop, and function, users familiar with Python can easily manipulate data using our DSL. Users can also implement any algorithm for configuring and modifying scene attributes. For ease of use, we provide several built-in samplers for users to easily generate diverse scenes for domain randomization.

In this section, we show some use cases by the built-in functions and samplers of our system. Due to the space limitations, we refer readers to our project page for more examples.

Basic usage. As our pipeline contains three different process stages, we provide corresponding processors, i.e., `SceneProcessor`, `EntityProcessor`, and `PixelProcessor` for users to manipulate data in each stage. To write DSL for a specific stage, the users need to declare a class inheriting from the corresponding built-in processor class. Then, the users need to write their customized operations in the `process` function.

Scene filtering. In the Scene Process Stage, users could use DSL to filter scenes according to their requirements. The following code shows how to select the scenes with more than two rooms, and each room area should be larger than 20m^2:

```
class SceneFilter(SceneProcessor):
    def process(self):
        if len(self.get_rooms) <= 2:
            self.filter_out()
        for room in self.get_rooms():
            if room.area < 20:
                self.filter_out()
```

We declare a class `SceneFilter` for the Scene Process Stage by inheriting the `SceneProcessor` class. Customized scene filtering operation are implemented in the `process` function.

Scene-level randomization. Users could rearrange the furniture layout for each room using our built-in scene-level sampler:

```
class FurnitureLayoutSampler(SceneProcessor):
    def process(self):
        for room in self.get_rooms():
            room.sample_layout()
```

Figure 3(a) shows the rearranged furniture layouts. Different from transforming each furniture randomly, our novel furniture layouts are reasonable because of considering multiple constraints from interior design guidelines.

Entity-level randomization. Users could easily sample attributes of entities, such as material, CAD model, and light. In the following example, we randomly replace the CAD models and materials in the “sofa” and “table” categories and randomize lighting using the entity-level sampler:

```
class EntitySampler(EntityProcessor):
    def process(self):
        for instance in self.shader.world.instances:
            if instance.label in [5, 8]:
                # 5: sofa, 8: table
                self.shader.world.replace_model(
                    id=instance.id)
                self.shader.world.replace_material(
                    id=instance.id)
        for light in self.shader.world.lights:
            light.tune_temp()
            light.tune_intensity()
```

Figure 3 (b-d) shows the results of different entity-level samplers. With these samplers, the users can improve the diversity of the scene. Except for randomization, some fine-grained operations like the transformation of each furniture or light intensity of each light can also be manually configured.

Non-imagery data. Our system also supports to export non-imagery data. We provide the function for users to export desired attributes easily. The following example shows the generation of camera information:

```
class CustomUserOutput(EntityProcessor):
    def process(self):
        for camera in self.shader.world.cameras:
            # export the position and target of camera
            self.shader.world.pick(
                type="camera", position=camera.position,
                target=camera.lookAt)
```

The `pick` method allows users to output any attributes as a JSON file. We utilize this feature to generate ground truths for the layout estimation task.

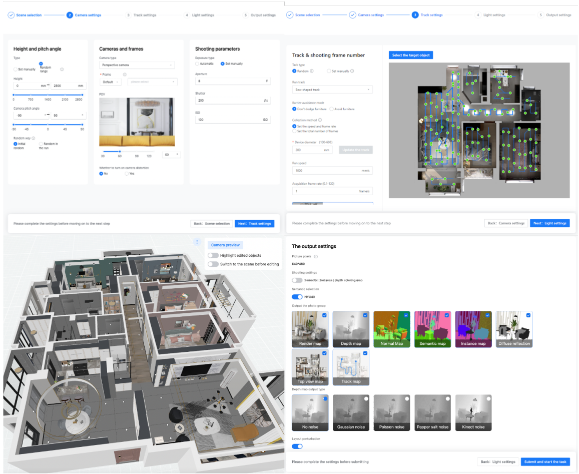
Pixel-level randomization. The users could apply customized computations to the pixels. Here, we use the built-in pixel-level sampler to add different types of noise to the images:

```
class DepthNoiseSampler(PixelProcessor):
    def process(self):
        # 4 represent Kinect noise model
        self.gen_depth(noise=4)
```

The results of multiple noise simulation are shown in Figure 4.

Trajectory generation. Finally, we show how to use our system to generate a video sequence by setting the camera trajectory:

```
class TrajectorySampler(EntityProcessor):
    def process(self):
        for camera in self.shader.world.cameras:
            camera.set_attr("imageWidth", 640)
            camera.set_attr("imageHeight", 480)
            self.shader.world.add_trajectory(
                id='traj_'+camera.id, initCamera=camera,
                fps=3, speed=1200, time=5, height=1000,
                collisionPadding=300, type="RANDOM")
```



✓ Scene selection — 2 Camera settings — 3 Track settings — 4 Light settings — 5 Output settings

Height and pitch angle

Type: ○ Set manually ● Random range

Height: 0 mm – 2800 mm

Camera pitch angle: -90 – 90

Random way: ● Initial random ○ Random in the run

Cameras and frames

Camera type: Perspective camera

* Frame: Default

FOV: 60

Whether to turn on camera distortion: ● No ○ Yes

Shooting parameters

Exposure type: ○ Automatic ● Set manually

Aperture: 8 F

Shutter: 200 /1s

ISO: 100 ISO

Please complete the settings before moving on to the next step [Back: Scene selection] [Next: Track settings]

✓ Scene selection — ✓ Camera settings — 3 Track settings — 4 Light settings — 5 Output settings

Track & shooting frame number

Tack type: ● Random ○ Set manually

Run track: Bow-shaped track

Barrier-avoidance mode: ● Don't dodge furniture ○ Avoid furniture

Collection method: ● Set the speed and frame rate ○ Set the total number of frames

* Device diameter (100-600): 200 mm [Update the track]

Run speed: 1000 mm/s

Acquisition frame rate (0.1-120): 1 frame/s

[Select the target object]

Please complete the settings before moving on to the next step [Back: Camera settings] [Next: Light settings]

Camera preview

Highlight edited objects

Switch to the scene before editing

The output settings

Picture pixels: 640*480

Shooting settings: Semantic | Instance | depth coloring map

Semantic selection: NYU40

Output the photo group: ☑ Render map ☑ Depth map ☑ Normal Map ☑ Semantic map ☑ Instance map ☑ Diffuse reflection ☑ Top view map ☑ Track map

Depth map output type: ● No noise ○ Gaussian noise ○ Poisson noise ○ Pepper salt noise ○ Kinect noise

Layout perturbation

Please complete the settings before submitting [Back: Light settings] [Submit and start the task]

Figure 5: *Screenshots of the GUI in our system. User-friendly GUI makes our system easier to use especially for novice users.*

We first set the image resolution. Then, we add a trajectory to the camera with customized speed and frame rate. The sampler will generate a random trajectory using the built-in algorithm [HPSC16]. Combining with scene layout information, the users can also generate camera trajectories by manually setting the key locations within the scene.

In our implementation, we employ the Entity Component System (ECS) architecture [GdAN14] to organize the scenes for customization features in our system. As commonly used in 3D game engines, the ECS architecture provides a flexible and intuitive way to describe and manipulate scenes compared with the object-oriented design. The ECS architecture treats each object in the scene as an entity. Each entity contains various components storing attributes, e.g., transformation, CAD model, material. Our system decouples data from logic by utilizing the ECS architecture, which empowers users to plug in customized computations with DSL to control data synthesis procedures. Furthermore, we integrate the randomization feature into the ECS architecture by attaching a distribution to depict each component. The newly proposed architecture is named ECS-D, where D denotes distributions on components.

Based on the Python programming language, ECS-D architecture, and built-in samplers, different data syntheses can be customized by users via DSL easily. In our implementation, we also provide a user-friendly GUI to represent some functions in our DSL for novice users. Such as scene filtering by specific constraints, camera settings, trajectory editor, and renderer configurations. Users can also check each modified scene using the 3D viewer. Figure 5 shows some screenshots of our GUI.

6. Experiments

To verify the usefulness of our system, we use three crucial interior scene understanding tasks, including room layout estimation, semantic segmentation, and depth estimation. We use the DSL to

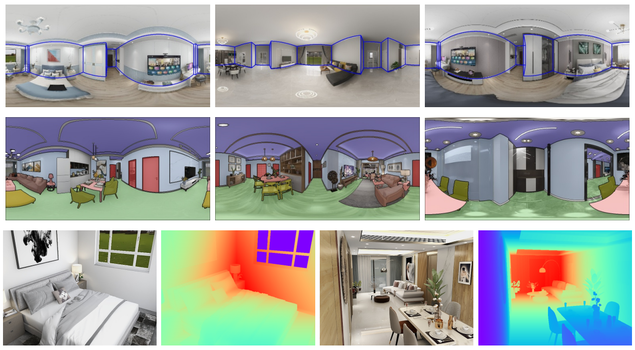


Figure 6: *Data samples generated by our system for Manhattan room layout estimation task (first row), semantic segmentation task (second row) and depth estimation task (third row).*

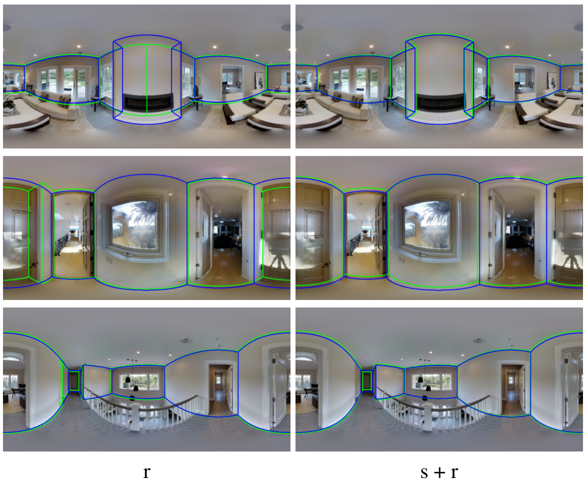


Figure 7: *Qualitative results of the Manhattan room layout estimation on the MatterportLayout dataset. Blue lines denote the ground-truth layout, while green lines denote the predicted layout.*

customize the pipeline for data synthesis. We demonstrate the capability of our system from two following perspectives: (1) Flexibility: our system has the potential in different vision tasks. (2) Validity: our system can boost the performance of existing methods using the synthesized data.

For each task, the models are trained using two following training strategies: (1) training only on the real dataset ("r") and (2) training on both synthetic and real dataset with Balanced Gradient Contribution (BGC) [RSAW16] ("s + r"). All experiments are conducted on two NVIDIA Tesla V100 GPUs with 32 GB memory.

Due to the space limitations, we refer readers to the supplementary material for the DSL codes and more quantitative and qualitative results.

6.1. Manhattan Room Layout Estimation

In this experiment, we first select scenes with Manhattan-world assumption in the Scene Process Stage. To increase the scene diver-

Table 2: *Manhattan room layout estimation results on Matterport-Layout dataset.*

Config.	#Data	3D IoU (%) ↑	2D IoU (%) ↑	RMSE ↓	δ_1 ↑
r	-	75.39	77.95	0.277	0.908
s + r	1.2K	76.56	79.22	0.275	0.912
s + r	12K	76.74	79.25	0.261	**0.919**
s + r	120K	**76.92**	**79.49**	**0.258**	0.918

Table 3: *Semantic segmentation results on 2D-3D-S dataset. DA: Domain Randomization. PA: Pixel Accuracy.*

Config.	#Scene	#Data	DA	Mean IoU (%) ↑	PA (%) ↑
r	-	-	-	47.08	80.02
s + r	6.7K	30K	✗	48.50	81.49
s + r	26.7K	120K	✗	**50.88**	82.34
s + r	6.7K	120K	✓	50.18	**82.57**

sity, we also filter out the room with less than four furniture objects. We randomly place a panoramic camera with the resolution of 1024 × 512 in each room in the Entity Process Stage. Finally, we export the positions of room corners and camera parameters. We synthesize 120K panorama images using our system. We use MatterportLayout [CDF*17, ZSP*21] as the real data. The MatterportLayout dataset consists of 1,647 images for training, 190 images for validation, and 458 images for testing.

We adopt HorizonNet [SHSC19] as the baseline approach. We use an Adam optimizer with an initial learning rate of 3×10^{-4} with a polynomial decay policy. For "r", we set the mini-batch size to 24. For "s + r", each batch contains 16 images from the real dataset and 8 from the synthetic dataset. For each strategy, we train the network for 30K iterations.

We adopt four standard metrics: 3D IoU, 2D IoU, root mean squared error (RMSE) and threshold accuracy δ_1 for evaluation. The results are shown in Table 2. As can be seen, the model trained on both the synthetic and real datasets achieves better results. Furthermore, the model with more synthetic data is better.

6.2. Semantic Segmentation

In this experiment, we filter out the rooms with less than four furniture objects. In the Entity Process Stage, a panoramic camera with the 1024 × 512 resolution is randomly placed in the room. In the Pixel Process Stage, we export semantic labels with NYUv2 40 label set [SHKF12]. We use our system to generate 120K panoramas. We use 1k images from 2D-3D-S [ASZS17] as the real data. Nine overlapping categories of the two datasets are used for evaluation: wall, floor, chair, sofa, door, window, bookshelf, ceiling, and table.

We use PSPNet [ZSQ*17] with dilated ResNet-50 as the backbone. We use an SGD optimizer with an initial learning rate of 2×10^{-2} with a polynomial decay policy, momentum 0.9, and weight decay of 10^{-4}. We set the mini-batch size to 8. In "s + r", each batch contains 4 images from the real dataset and 4 from the synthetic dataset. For each strategy, we train the whole network for 20K iterations.

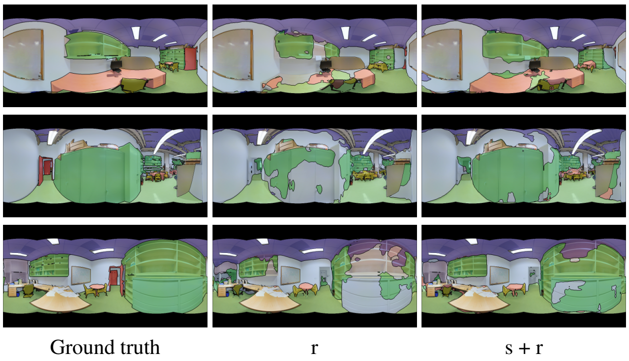


Figure 8: *Semantic segmentation results of PSPNet. Different colors denote different semantic categories.*

Table 4: *Depth estimation results on NYUv2 dataset.*

Methods	Config.	Rel ↓	$\log_{10}$ ↓	RMSE ↓	δ_1 ↑	δ_2 ↑	δ_3 ↑
VNL [YLSY19]	r	0.133	0.057	0.573	0.833	0.969	0.991
	s + r	**0.116**	**0.050**	**0.528**	**0.869**	**0.976**	**0.994**
AdaBins [BAW21]	r	0.134	0.055	0.452	0.841	0.972	0.992
	s + r	**0.129**	**0.053**	**0.421**	**0.856**	**0.975**	**0.995**

We adopt two standard metrics: Mean IoU and Pixel Accuracy (PA). The results are reported in Table 3. Training with our synthetic dataset achieves better performance. The performance can be further improved when using more scenes. Furthermore, we conduct an ablation study on Domain Randomization (DR) of layout, camera, light, model, and material. We generate more images (120K) with limited scenes (6.7K) by incorporating DR technique. When using the same number of scenes, the result with DR (the fourth row) is better than that without DR (the second row). The result with generated scenes by DR is comparable with that using more original designed scenes (the third row). This verifies the effectiveness of DR.

6.3. Depth Estimation

In this experiment, the camera are placed by customized heuristic rules. We set the image resolution as 640 × 480 and the horizontal field-of-view (FoV) to 57 degree in the Entity Process Stage. The depth images are exported in the Pixel Process Stage. We use our system to generate 120k data. We use 1449 images from labeled NYUv2 dataset [SHKF12] as the real data.

We adopt two methods for this task: VNL [YLSY19] and AdaBins [BAW21]. For VNL [YLSY19], we use an SGD optimizer with an initial learning rate 1×10^{-4} with polynomial decay policy, momentum 0.9, and weight decay 5×10^{-4}. We set the mini-batch size to 8. In "s + r", each batch contains 4 images from the real dataset and 4 from the synthetic dataset. For AdaBins [BAW21], we use an AdamW optimizer with max learning rate 0.000357 with 1-cycle learning rate decay policy [ST19], and weight decay 0.1. We set the mini-batch size to 16. In "s + r", each batch contains 10 images from the real dataset and 6 from the synthetic dataset. For

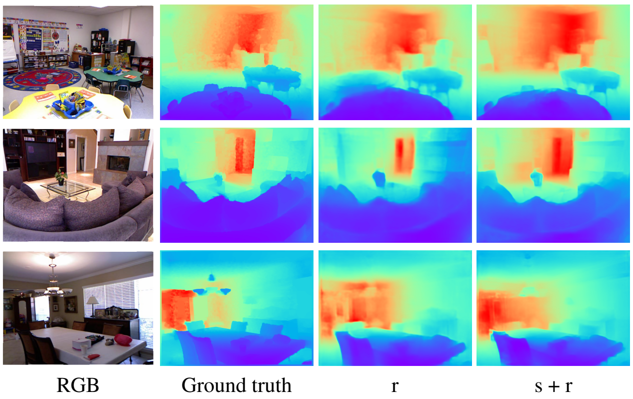


Figure 9: *Qualitative results of the depth estimation with VNL [YLSY19].*

Table 5: *Data generalization results on the DIODE dataset [VKZ*19]. The best and the second best results are boldfaced and underlined.*

Train Set	#Data	Rel ↓	$\log_{10}$ ↓	RMSE ↓	δ_1 ↑	δ_2 ↑	δ_3 ↑
NYUv2	795	**0.463**	0.297	2.600	0.181	0.409	0.605
Ours	795	0.509	0.255	**2.386**	0.283	0.517	0.666
Ours	120K	0.503	**0.249**	2.513	**0.359**	**0.580**	**0.712**

each strategy in both methods, we train the whole network for 40K iterations.

To perform a reasonable evaluation, we use four common metrics: average absolute relative error (Rel), average $\log_{10}$ error ($\log_{10}$), room mean squared error (RMSE) and threshold accuracy ($\delta_i < 1.25^i$, where $i = 1, 2, 3$). Table 4 summarizes the quantitative results. It can be seen that augmenting real datasets with our synthetic data achieves better performances for both methods.

Data generalization. We further conduct an experiment to compare the generalization of the models trained on our synthetic dataset or the real dataset. We use an accurate and diverse real-world captured depth dataset, DIODE dataset [VKZ*19], as the test dataset. We use the NYUv2 dataset [SHKF12] and our synthetic dataset as training data separately. In this experiment, we use AdaBins [BAW21] as the baseline approach.

The results are shown in Table 5. As one can see, the model trained on our synthetic data has better generalization performance than that trained on NYUv2 [SHKF12], even with the same number of training samples. This may be due to the limited depth ranges in the NYUv2 dataset. In contrast, our dataset and DIODE dataset has more diverse depth samples. When using more training data from our dataset, the performance is significantly improved.

7. Conclusion

This paper presents MINERVAS, a programmable system for interior imagery data synthesis. The system employs ECS-D architecture for scene representation and introduces an easy-to-use and flexible DSL for task-specific customization. Additionally, we introduce multi-level samplers for randomization to increase the diversity of the synthetic data. The experimental results validate the ability of our system to boost the performance on various interior scene understanding tasks.

Limitations and future work. In order to protect the copyright of 3D assets, explicit 3D mesh data can not be obtained from our system. Thus our system can not facilitate some tasks operating on the mesh directly. As our system can generate any geometry cues in the image space, our system can still facilitate a broad range of applications in scene understanding or scene synthesis. Since we mainly focus on computer vision tasks, our system currently does not support physics and interaction with virtual scenes. In the future, we plan to integrate the physics and real-time interactive simulation into the MINERVAS system for online embodied agent learning. Another promising direction for future work would be how to automatically synthesize data to minimize domain gaps between synthetic data and real-world data. Though our system protects 3D assets via remote rendering, the users might illegally reconstruct the 3D models using recent 3D reconstruction techniques. It is important to incorporate some advanced techniques of preventing reconstruction [KTL*04, KL05] in the future. There may exist several ways to further protect the copyright in our system. First, we can detect suspicious sequences or frequencies of image requests [KL05] in the system. Secondly, some watermarks (e.g., subtle noise) can be added to rendered images to prevent the accurate reconstruction [KL05]. The adversarial attack technique [GSS15] may also help this. Finally, the blockchain technique [PHY21] may also be applied to help identify the illegal release to protect the copyright.

Acknowledgments

This work was supported in part by NSFC (No. 61872319), Key R&D Program of Zhejiang Province (No. 2022C01025), and the Fundamental Research Funds for the Central Universities. We would like to thank Manycore Tech Inc. (Coohom & Kujiale) for providing the 3D indoor scene database and the high-performance rendering platform, especially Coohom Cloud Team for technical support.

References

[ADD*19] Avetisyan A., Dahnert M., Dai A., Savva M., Chang A. X., Niessner M.: Scan2cad: Learning cad model alignment in rgb-d scans. In *CVPR* (2019), pp. 2614–2623. 2

[ASZ*16] Armeni I., Sener O., Zamir A. R., Jiang H., Brilakis I., Fischer M., Savarese S.: 3d semantic parsing of large-scale indoor spaces. In *CVPR* (2016), pp. 1534–1543. 2

[ASZS17] Armeni I., Sax S., Zamir A. R., Savarese S.: Joint 2d-3d-semantic data for indoor scene understanding. *CoRR abs/1702.01105* (2017). 2, 9

[BAW21] Bhat S. F., Alhashim I., Wonka P.: Adabins: Depth estimation using adaptive bins. In *CVPR* (2021), pp. 4009–4018. 9, 10

[BCD*21] Borkman S., Crespi A., Dhakad S., Ganguly S., Hogins J., Jhang Y.-C., Kamalzadeh M., Li B., Leal S., Parisi P., Romero C., Smith W., Thaman A., Warren S., Yadav N.: Unity perception: Generate synthetic data for computer vision. *CoRR abs/2107.04259* (2021). 3

[BDK*18] Bondi E., Dey D., Kapoor A., Piavis J., Shah S., Fang F., Dilkina B., Hannaford R., Iyer A., Joppa L., et al.: Airsim-w: A simulation environment for wildlife conservation with uavs. In *COMPASS* (2018), pp. 1–12. 3

[BM13] Barron J. T., Malik J.: Intrinsic scene properties from a single rgb-d image. In *CVPR* (2013), pp. 17–24. 7

[BRHS14] Bohg J., Romero J., Herzog A., Schaal S.: Robot arm pose estimation through pixel-wise part classification. In *ICRA* (2014), pp. 3143–3150. 7

[CDF*17] Chang A. X., Dai A., Funkhouser T. A., Halber M., Niessner M., Savva M., Song S., Zeng A., Zhang Y.: Matterport3d: Learning from RGB-D data in indoor environments. In *3DV* (2017), pp. 667–676. 1, 2, 3, 9

[CSC*20] Chou S.-H., Sun C., Chang W.-Y., Hsu W.-T., Sun M., Fu J.: 360-indoor: Towards learning real-world objects in 360deg indoor equirectangular images. In *WACV* (2020), pp. 845–853. 1, 2

[DCS*17] Dai A., Chang A. X., Savva M., Halber M., Funkhouser T., Niessner M.: Scannet: Richly-annotated 3d reconstructions of indoor scenes. In *CVPR* (2017), pp. 5828–5839. 1, 2

[DRC*17] Dosovitskiy A., Ros G., Codevilla F., Lopez A., Koltun V.: Carla: An open urban driving simulator. In *CoRL* (2017), vol. 78, pp. 1–16. 3

[DSW*19] Denninger M., Sundermeyer M., Winkelbauer D., Zidan Y., Olefir D., Elbadrawy M., Lodhi A., Katam H.: Blenderproc. *CoRR 1911.01911* (2019). 3

[ESMZ21] Eftekhar A., Sax A., Malik J., Zamir A.: Omnidata: A scalable pipeline for making multi-task mid-level vision datasets from 3d scans. In *ICCV* (2021), pp. 10786–10796. 3

[FCG*21] Fu H., Cai B., Gao L., Zhang L., Li C., Zeng Q., Sun C., Fei Y., Zheng Y., Li Y., Liu Y., Liu P., Ma L., Weng L., Hu X., Ma X., Qian Q., Jia R., Zhao B., Zhang H.: 3d-front: 3d furnished rooms with layouts and semantics. In *CVPR* (2021), pp. 10933–10942. 1, 2

[FDG*19] Fremont D. J., Dreossi T., Ghosh S., Yue X., Sangiovanni-Vincentelli A. L., Seshia S. A.: Scenic: a language for scenario specification and scene generation. In *PLDI* (2019), pp. 63–78. 3

[FJG*21] Fu H., Jia R., Gao L., Gong M., Zhao B., Maybank S., Tao D.: 3d-future: 3d furniture shape with texture. *IJCV 129* (2021), 3313–3337. 2

[FRS*12] Fisher M., Ritchie D., Savva M., Funkhouser T., Hanrahan P.: Example-based synthesis of 3d object arrangements. *ACM TOG 31*, 6 (2012), 1–11. 2

[GBB*22] Greff K., Belletti F., Beyer L., Doersch C., Du Y., Duckworth D., Fleet D. J., Gnanapragasam D., Golemo F., Herrmann C., Kipf T., Kundu A., Lagun D., Laradji I., Liu H.-T. D., Meyer H., Miao Y., Nowrouzezahrai D., Oztireli C., Pot E., Radwan N., Rebain D., Sabour S., Sajjadi M. S. M., Sela M., Sitzmann V., Stone A., Sun D., Vora S., Wang Z., Wu T., Yi K. M., Zhong F., Tagliasacchi A.: Kubric: A scalable dataset generator. In *CVPR* (2022), pp. 3749–3761. 3

[GdAN14] Garcia F. E., de Almeida Neris V. P.: A data-driven entity-component approach to develop universally accessible games. In *UAHCI* (2014), pp. 537–548. 8

[GGMGO*18] Garcia-Garcia A., Martinez-Gonzalez P., Oprea S., Castro-Vargas J. A., Orts-Escolano S., Garcia-Rodriguez J., Jover-Alvarez A.: The robotrix: An extremely photorealistic and very-large-scale indoor dataset of sequences with robot trajectories and interactions. In *IROS* (2018), pp. 6790–6797. 2

[GS14] Goodman N. D., Stuhlmüller A.: The design and implementation of probabilistic programming languages. http://dippl.org, 2014. Accessed: 2021-10-7. 3

[GSA*21] Gan C., Schwartz J., Alter S., Mrowca D., Schrimpf M., Traer J., De Freitas J., Kubilius J., Bhandwaldar A., Haber N., Sano M., Kim K., Wang E., Lingelbach M., Curtis A., Feigelis K., Bear D., Gutfreund D., Cox D., Torralba A., DiCarlo J. J., Tenenbaum J., McDermott J., Yamins D.: Threedworld: A platform for interactive multi-modal physical simulation. In *NeurIPS Datasets and Benchmarks Track* (2021). 3

[GSS15] Goodfellow I. J., Shlens J., Szegedy C.: Explaining and harnessing adversarial examples. In *ICLR* (2015). 10

[HPN*16] Hua B.-S., Pham Q.-H., Nguyen D. T., Tran M.-K., Yu L.-F., Yeung S.-K.: Scenenn: A scene meshes dataset with annotations. In *3DV* (2016), pp. 92–101. 2

[HPSC16] Handa A., Pătrăucean V., Stent S., Cipolla R.: Scenenet: An annotated model generator for indoor scene understanding. In *ICRA* (2016), pp. 5737–5743. 1, 2, 6, 8

[HWMD14] Handa A., Whelan T., McDonald J., Davison A. J.: A benchmark for rgb-d visual odometry, 3d reconstruction and slam. In *ICRA* (2014), pp. 1524–1531. 7

[JHHB16] Johnson M., Hofmann K., Hutton T., Bignell D.: The malmo platform for artificial intelligence experimentation. In *IJCAI* (2016), pp. 4246–4247. 3

[JKJ*11] Janoch A., Karayev S., Jia Y., Barron J. T., Fritz M., Saenko K., Darrell T.: A category-level 3d object dataset: Putting the kinect to work. In *ICCV Workshop* (2011), pp. 1168–1174. 2

[JQZ*18] Jiang C., Qi S., Zhu Y., Huang S., Lin J., Yu L.-F., Terzopoulos D., Zhu S.-C.: Configurable 3d scene synthesis and 2d image rendering with per-pixel ground truth using stochastic grammars. *IJCV 126*, 9 (2018), 920–941. 3

[KAJS11] Koppula H. S., Anand A., Joachims T., Saxena A.: Semantic labeling of 3d point clouds for indoor scenes. In *NeurIPS* (2011), pp. 244–252. 2

[KK18] Kán P., Kaufmann H.: Automatic furniture arrangement using greedy cost minimization. In *VR* (2018), pp. 491–498. 3, 5

[KKTM15] Kulkarni T. D., Kohli P., Tenenbaum J. B., Mansinghka V.: Picture: A probabilistic programming language for scene perception. In *CVPR* (2015), pp. 4390–4399. 3

[KL05] Koller D., Levoy M.: Protecting 3d graphics content. *Commun. ACM 48*, 6 (2005), 74–80. 3, 5, 10

[KMH*17] Kolve E., Mottaghi R., Han W., VanderBilt E., Weihs L., Herrasti A., Gordon D., Zhu Y., Gupta A., Farhadi A.: Ai2-thor: An interactive 3d environment for visual ai. *CoRR abs/1712.05474* (2017). 2

[KTL*04] Koller D., Turitzin M., Levoy M., Tarini M., Croccia G., Cignoni P., Scopigno R.: Protected interactive 3d graphics via remote rendering. *ACM TOG 23*, 3 (2004), 695–703. 3, 10

[KWR*16] Kempka M., Wydmuch M., Runc G., Toczek J., Jaśkowski W.: Vizdoom: A doom-based ai research platform for visual reinforcement learning. In *CIG* (2016), pp. 1–8. 3

[LBF14] Lai K., Bo L., Fox D.: Unsupervised feature learning for 3d scene labeling. In *ICRA* (2014), pp. 3050–3057. 2

[LPX*19] Li M., Patil A. G., Xu K., Chaudhuri S., Khan O., Shamir A., Tu C., Chen B., Cohen-Or D., Zhang H.: Grains: Generative recursive autoencoders for indoor scenes. *ACM TOG 38*, 2 (2019), 1–16. 3

[LS18] Li Z., Snavely N.: Cgintrinsics: Better intrinsic image decomposition through physically-based rendering. In *ECCV* (2018), pp. 371–387. 1, 2

[LSM*18] Li W., Saeedi S., McCormac J., Clark R., Tzoumanikas D., Ye Q., Huang Y., Tang R., Leutenegger S.: Interiornet: Mega-scale multi-sensor photo-realistic indoor scenes dataset. In *BMVC* (2018). 1, 2

[LYS*21] Li Z., Yu T.-W., Sang S., Wang S., Bi S., Xu Z., Yu H.-X., Sunkavalli K., Hašan M., Ramamoorthi R., et al.: Openrooms: An end-to-end open framework for photorealistic indoor scene datasets. In *CVPR* (2021). 1, 2

[LZZ*19] Liu B., Zhang J., Zhang X., Zhang W., Yu C., Zhou Y.: Furnishing your room by what you see: An end-to-end furniture set retrieval framework with rich annotated benchmark dataset. *CoRR abs/1911.09299* (2019). 2

[MHLD17] McCormac J., Handa A., Leutenegger S., Davison A. J.: Scenenet rgb-d: Can 5m synthetic images beat generic imagenet pre-training on indoor segmentation? In *ICCV* (2017), pp. 2678–2687. 1, 2

[MSL*11] Merrell P., Schkufza E., Li Z., Agrawala M., Koltun V.: Interactive furniture layout using interior design guidelines. *ACM TOG 30*, 4 (2011), 1–10. 3

[MTL*21] Morrical N., Tremblay J., Lin Y., Tyree S., Birchfield S., Pascucci V., Wald I.: Nvisii: A scriptable tool for photorealistic image generation. In *ICLR Workshop* (2021). 3

[PHY21] Park H., Huo Y., Yoon S.-E.: Meshchain: Secure 3d model and intellectual property management powered by blockchain technology. In *CGI* (2021), pp. 519–534. 3, 10

[RGW*21] Ramakrishnan S. K., Gokaslan A., Wijmans E., Maksymets O., Clegg A., Turner J. M., Undersander E., Galuba W., Westbury A., Chang A. X., Savva M., Zhao Y., Batra D.: Habitat-matterport 3d dataset (HM3d): 1000 large-scale 3d environments for embodied AI. In *NeurIPS Datasets and Benchmarks Track* (2021). 2

[RP21] Roberts M., Paczan N.: Hypersim: A photorealistic synthetic dataset for holistic indoor scene understanding. In *ICCV* (2021), pp. 10912–10922. 1, 2

[RSAW16] Ros G., Stent S., Alcantarilla P. F., Watanabe T.: Training constrained deconvolutional networks for road scene semantic segmentation. *CoRR abs/1604.01545* (2016). 8

[RVRK16] Richter S. R., Vineet V., Roth S., Koltun V.: Playing for data: Ground truth from computer games. In *ECCV* (2016), pp. 102–118. 3

[RWL19] Ritchie D., Wang K., Lin Y.-a.: Fast and flexible indoor scene synthesis via deep convolutional generative models. In *CVPR* (2019), pp. 6182–6190. 3

[SCH*16] Savva M., Chang A. X., Hanrahan P., Fisher M., Niessner M.: Pigraphs: learning interaction snapshots from observations. *ACM TOG 35*, 4 (2016), 1–12. 2

[SEE*12] Sturm J., Engelhard N., Endres F., Burgard W., Cremers D.: A benchmark for the evaluation of rgb-d slam systems. In *IROS* (2012), pp. 573–580. 2

[SHKF12] Silberman N., Hoiem D., Kohli P., Fergus R.: Indoor segmentation and support inference from rgbd images. In *ECCV* (2012), pp. 746–760. 1, 2, 4, 9, 10

[SHSC19] Sun C., Hsiao C.-W., Sun M., Chen H.-T.: Horizonnet: Learning room layout with 1d representation and pano stretch data augmentation. In *CVPR* (2019), pp. 1047–1056. 9

[SKM*19] Savva M., Kadian A., Maksymets O., Zhao Y., Wijmans E., Jain B., Straub J., Liu J., Koltun V., Malik J., et al.: Habitat: A platform for embodied ai research. In *ICCV* (2019), pp. 9339–9347. 3

[SLX15] Song S., Lichtenberg S. P., Xiao J.: Sun rgb-d: A rgb-d scene understanding benchmark suite. In *CVPR* (2015), pp. 567–576. 1, 2

[ST19] Smith L. N., Topin N.: Super-convergence: Very fast training of neural networks using large learning rates. In *SPIE* (2019), vol. 11006, p. 1100612. 9

[SWM*19] Straub J., Whelan T., Ma L., Chen Y., Wijmans E., Green S., Engel J. J., Mur-Artal R., Ren C., Verma S., Clarkson A., Yan M., Budge B., Yan Y., Pan X., Yon J., Zou Y., Leon K., Carter N., Briales J., Gillingham T., Mueggler E., Pesqueira L., Savva M., Batra D., Strasdat H. M., Nardi R. D., Goesele M., Lovegrove S., Newcombe R.: The Replica dataset: A digital replica of indoor spaces. *CoRR abs/1906.05797* (2019). 2, 3

[SYZ*17] Song S., Yu F., Zeng A., Chang A. X., Savva M., Funkhouser T.: Semantic scene completion from a single depth image. In *CVPR* (2017), pp. 1746–1754. 1, 2

[SZ15] Simonyan K., Zisserman A.: Very deep convolutional networks for large-scale image recognition. In *ICLR* (2015). 6

[TFR*17] Tobin J., Fong R., Ray A., Schneider J., Zaremba W., Abbeel P.: Domain randomization for transferring deep neural networks from simulation to the real world. In *IROS* (2017), pp. 23–30. 4

[VKZ*19] Vasiljevic I., Kolkin N., Zhang S., Luo R., Wang H., Dai F. Z., Daniele A. F., Mostajabi M., Basart S., Walter M. R., Shakhnarovich G.: DIODE: A Dense Indoor and Outdoor DEpth Dataset. *CoRR abs/1908.00463* (2019). 10

[WLW*19] Wang K., Lin Y.-A., Weissmann B., Savva M., Chang A. X., Ritchie D.: Planit: Planning and instantiating indoor scenes with relation graph and spatial prior networks. *ACM TOG 38*, 4 (2019), 1–15. 3

[WWGT18] Wu Y., Wu Y., Gkioxari G., Tian Y.: Building generalizable agents with a realistic and rich 3d environment. In *ICLR Workshop* (2018). 2

[WZW*20] Wang W., Zhu D., Wang X., Hu Y., Qiu Y., Wang C., Hu Y., Kapoor A., Scherer S.: Tartanair: A dataset to push the limits of visual slam. In *IROS* (2020), pp. 4909–4916. 2

[XOT13] Xiao J., Owens A., Torralba A.: Sun3d: A database of big spaces reconstructed using sfm and object labels. In *ICCV* (2013), pp. 1625–1632. 1, 2

[XZH*18] Xia F., Zamir A. R., He Z., Sax A., Malik J., Savarese S.: Gibson env: Real-world perception for embodied agents. In *CVPR* (2018), pp. 9068–9079. 1, 2, 3

[YLSY19] Yin W., Liu Y., Shen C., Yan Y.: Enforcing geometric constraints of virtual normal for depth prediction. In *ICCV* (2019), pp. 5684–5693. 9, 10

[YYT15] Yu L.-F., Yeung S.-K., Terzopoulos D.: The clutterpalette: An interactive tool for detailing indoor scenes. *IEEE TVCG 22*, 2 (2015), 1138–1148. 3

[ZSP*21] Zou C., Su J., Peng C., Colburn A., Shan Q., Wonka P., Chu H., Hoiem D.: Manhattan room layout reconstruction from a single 360 image: A comparative study of state-of-the-art methods. *IJCV 129*, 5 (2021), 1410–1431. 9

[ZSQ*17] Zhao H., Shi J., Qi X., Wang X., Jia J.: Pyramid scene parsing network. In *CVPR* (2017), pp. 2881–2890. 9

[ZSY*17] Zhang Y., Song S., Yumer E., Savva M., Lee J.-Y., Jin H., Funkhouser T.: Physically-based rendering for indoor scene understanding using convolutional neural networks. In *CVPR* (2017), pp. 5287–5295. 2, 4

[ZXZ21] Zhang S.-K., Xie W.-Y., Zhang S.-H.: Geometry-based layout generation with hyper-relations among objects. *Graphical Models 116* (2021), 101104. 3

[ZYM*20] Zhang Z., Yang Z., Ma C., Luo L., Huth A., Vouga E., Huang Q.: Deep generative modeling for scene synthesis via hybrid representations. *ACM TOG 39*, 2 (2020), 1–21. 3

[ZZL*20] Zheng J., Zhang J., Li J., Tang R., Gao S., Zhou Z.: Structured3d: A large photo-realistic dataset for structured 3d modeling. In *ECCV* (2020), pp. 519–535. 1, 2, 6

[ZZX*21] Zhang S.-H., Zhang S.-K., Xie W.-Y., Luo C.-Y., Yang Y., Fu H.: Fast 3d indoor scene synthesis by learning spatial relation priors of objects. *IEEE TVCG* (2021). 3