Pacific Graphics 2022
N. Umetani, E. Vouga, and C. Wojtan
(Guest Editors)

Volume 40 (2021), Number 7

Supplementary Material for 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.

In this supplementary material, we first show all experimental details, including the DSL codes for generating data, and additional results. Then, we provide additional examples of random samplers used in our system.

1. Experimental Details

1.1. Manhattan Room Layout Estimation

DSL code. Code 1 shows the DSL code for this task. We first filter scenes with Manhattan-world assumption using `ManhattanSceneFilter` in the Scene Process Stage. In the Entity Process Stage, we then remove the cameras in the relatively empty or non-Manhattan rooms using `CameraFilter`. Next, we set the camera parameters (e.g., camera type and image resolution) using `CameraSetting`. We randomize the positions of the cameras in `CameraRandomizer`. After the Rendering Process Stage, we record the positions of room corners and camera parameters using `StructureOutput`.

Quantitative results. The detailed results are shown in Table 1. As can be seen, the model trained on both the synthetic and real datasets achieves better result than that trained on real dataset only. We also conduct experiments using different synthetic data size. The result with largest synthetic data size is the best as all of the metrics are the best or second best result.

1.2. Semantic Segmentation

DSL code. Code 2 shows the DSL for this task. In the Entity Process Stage, we first filter out cameras in the rooms with less than four pieces of furniture using `CameraFilter`. For each camera, we set the camera model as "panorama" and the resolution as 1024×512 in `CameraSetting`. Then, we utilize the built-in domain randomization technique of the system to generate more data. Specifically, we randomize room layout in the Scene Process Stage using `FurnitureLayoutSampler`. We randomize material, model, and camera in the Entity Process Stage using `EntityRandomizer`. In the Pixel Process Stage, we generate semantic labels. In order to align with the semantic labels in real dataset, we modify the label of NYUv2 40 label set as shown in `SemanticOutput` class.

1.3. Depth Estimation

DSL code. Code 3 shows the DSL for this task. In the Entity Process Stage, we manually setup cameras in the valid rooms as shown in class `CameraSetter`. Specifically, we set the camera positions by some heuristic rules, the image resolution as 640×480, and horizontal field-of-view (FoV) to $57°$ to align with Microsoft Kinect used in the NYUv2 dataset. In the Pixel Process Stage, we setup the depth output in class `DepthOutput`.

Qualitative results. We show qualitative results in Figure 1. As one can see, training on the synthetic and real dataset, the network generates more accurate estimations than that only using real images for training for both methods. The noise has been significantly reduced in the depth estimation. Note that the ground-truth depth maps used by AdaBins [BAW21] have some missing depth values because they use the raw depth data in the code.

2. Additional Results of Random Samplers

Furniture layout sampler. With the layout sampler, users can easily change the furniture arrangement in the Scene Process Stage, which could increase the diversity of the scene. We show the code example using the furniture layout sampler and then give more results in Figure 2.

Light sampler. With the light sampler, users could randomly change the light in the scene to simulate various lighting conditions. We show the code example using the light sampler and then give more results in Figure 3.

Model sampler. With the model sampler, users could replace the original model with a new model with an appropriate size and similar category. We show the code example using the model sampler and then give more results in Figure 4.

Material sampler. With the material sampler, users could sample the appropriate new material based on the original material type, according to the material information stored in the database. We show the code example using the material sampler and then give more results in Figure 5.

Custom sampler. The flexibility of our DSL allows users to create their customized sampling strategy. Code 4 shows an example of customized trajectory sampler DSL. We can further use this customized sampler for vision tasks, e.g., SLAM as in Figure 6.

Table 1: Detailed Manhattan layout estimation results on the MatterportLayout dataset. The best and the second best results are boldfaced and underlined.

Metric: 3D IoU (%) ↑						
Training strategies	Synthetic data size	Overall	4 corners	6 corners	8 corners	$\geq$ 10 corners
r	-	75.39	76.58	80.42	71.74	65.09
s + r	1.2K	76.56	78.03	**81.64**	72.16	65.70
s + r	12K	76.74	78.07	81.24	**73.39**	66.25
s + r	120K	**76.92**	**78.43**	81.39	72.43	**66.88**

Metric: 2D IoU (%) ↑						
Training strategies	Synthetic data size	Overall	4 corners	6 corners	8 corners	$\geq$ 10 corners
r	-	77.95	79.35	82.88	73.99	67.10
s + r	1.2K	79.22	80.95	**84.08**	74.56	67.56
s + r	12K	79.25	80.86	83.71	**75.52**	67.81
s + r	120K	**79.49**	**81.28**	83.94	74.63	**68.50**

Metric: RMSE ↓						
Training strategies	Synthetic data size	Overall	4 corners	6 corners	8 corners	$\geq$ 10 corners
r	-	0.277	0.246	0.218	0.318	0.492
s + r	1.2K	0.275	0.241	**0.207**	0.307	0.534
s + r	12K	0.261	**0.227**	0.214	0.304	0.469
s + r	120K	**0.258**	0.228	0.214	**0.299**	**0.444**

Metric: δ_1 ↑						
Training strategies	Synthetic data size	Overall	4 corners	6 corners	8 corners	$\geq$ 10 corners
r	-	0.908	0.900	0.947	0.914	0.875
s + r	1.2K	0.912	0.900	0.944	**0.933**	0.891
s + r	12K	**0.919**	**0.913**	**0.952**	0.922	0.892
s + r	120K	0.918	0.909	0.947	0.923	**0.912**

References

[BAW21] Bhat S. F., Alhashim I., Wonka P.: Adabins: Depth estimation using adaptive bins. In *CVPR* (2021), pp. 4009–4018. 1, 8

[DNZ*17] Dai A., Niessner M., Zollhöfer M., Izadi S., Theobalt C.: Bundlefusion: Real-time globally consistent 3d reconstruction using on-the-fly surface reintegration. *ACM TOG 36*, 4 (2017), 1. 14

```python
import numpy as np
from shapely.geometry import Point
import pandas as pd

from ksecs.resources import NYU40_MAPPING
from ksecs.ECS.processors.entity_processor import EntityProcessor
from ksecs.ECS.processors.scene_processor import SceneProcessor
from ksecs.ECS.processors.render_processor import RenderProcessor
from ksecs.ECS.processors.pixel_processor import PixelProcessor


class CameraFilter(EntityProcessor):
    def is_valid_room(self, room, num_furniture):
        # Check if the number of furniture in the room is above threshold.
        polygon = room.gen_polygon()
        count = 0
        for ins in self.shader.world.instances:
            if not ins.type == 'ASSET':
                continue
            if polygon.contains(Point([ins.transform[i] for i in [3, 7, 11]])):
                count += 1
        return count >= num_furniture

    def delete_cameras_in_room(self, room):
        polygon = room.gen_polygon()
        for camera in self.shader.world.cameras:
            if polygon.contains(Point([camera.position[axis] for axis in "xyz"])):
                self.shader.world.delete_entity(camera)

    def process(self):
        # We only use rooms with more than 4 assets
        for room in self.shader.world.rooms:
            if not self.is_valid_room(room, 4):
                self.delete_cameras_in_room(room)


class CameraSetting(EntityProcessor):
    def process(self):
        for camera in self.shader.world.cameras:
            camera.set_attr("imageWidth", 1024)
            camera.set_attr("imageHeight", 512)
            camera.set_attr("cameraType", "PANORAMA")


class FurnitureLayoutSampler(SceneProcessor):
    # Randomize layout
    def process(self):
        for room in self.shader.world.rooms:
            room.randomize_layout(self.shader.world)


class EntityRandomizer(EntityProcessor):
    def randomize_model_material(self):
        for instance in self.shader.world.instances:
            # Randomize material
            self.shader.world.replace_material(
                id=instance.id,
                type='REPLACE_ALL'
            )
            if instance.type == 'ASSET':
                # Randomize model
                self.shader.world.replace_model(
                    id=instance.id
                )

    def randomize_light(self):
        for light in self.shader.world.lights:
            # Randonly adjust the color temperature
            light._tune_temp(1)
            # Randomly adjust the light intensity
            light.tune_random(1.2)

    def randomize_camera(self):
        for camera in self.shader.world.cameras:
            random_vec = np.random.normal(0, 1, size=3)
            camera_pos = np.array(list(camera.position.values()))
            randomized_pos = camera_pos + random_vec * \
                np.array([500.0, 500.0, 100.0])
            camera.set_attr(
                'position', x=randomized_pos[0], y=randomized_pos[1], z=randomized_pos[2])
            camera.set_attr('lookAt', z=randomized_pos[2])

    def process(self):
        self.randomize_model_material()
        self.randomize_light()
        self.randomize_camera()


class Render(RenderProcessor):
    def process(self, *args, **kwargs):
        self.gen_rgb(distort=0, noise=0)


class SemanticOutput(PixelProcessor):
    def modify_nyu40_labels(self):
        # modify NYU40 labels for aligning with label set of '2D-3D-S' dataset
        s_map = pd.read_csv(NYU40_MAPPING)
        s_map = s_map[["id", "nyu_label_id"]]
        s_map.index = s_map["id"]
        s_map.drop(columns="id", inplace=True)
        s_map.loc[
            ~s_map["nyu_label_id"].isin(
                [1, 2, 5, 6, 7, 8, 9, 10, 14, 15, 22]
            ),
            "nyu_label_id"
        ] = 0
        s_map.loc[s_map["nyu_label_id"] == 5, "nyu_label_id"] = 3
        s_map.loc[s_map["nyu_label_id"] == 6, "nyu_label_id"] = 4
        s_map.loc[s_map["nyu_label_id"] == 8, "nyu_label_id"] = 5
        s_map.loc[s_map["nyu_label_id"] == 9, "nyu_label_id"] = 6
        s_map.loc[(s_map["nyu_label_id"] == 7) | (s_map["nyu_label_id"] == 14), "nyu_label_id"] = 9
        s_map.loc[(s_map["nyu_label_id"] == 10) | (s_map["nyu_label_id"] == 15), "nyu_label_id"] = 7
        s_map.loc[s_map["nyu_label_id"] == 22, "nyu_label_id"] = 8
        s_map = s_map.to_dict()["nyu_label_id"]
        return s_map

    def process(self, **kwargs):
        self.gen_semantic(label_arch=self.modify_nyu40_labels())
```

Code 1: DSL code for Manhattan room layout estimation task.

```python
import numpy as np
from shapely.geometry import Point
import pandas as pd

from ksecs.resources import NYU40_MAPPING
from ksecs.ECS.processors.entity_processor import EntityProcessor
from ksecs.ECS.processors.scene_processor import SceneProcessor
from ksecs.ECS.processors.render_processor import RenderProcessor
from ksecs.ECS.processors.pixel_processor import PixelProcessor


class CameraFilter(EntityProcessor):
    def is_valid_room(self, room, num_furniture):
        # Check if the number of furniture in the room is above threshold.
        polygon = room.gen_polygon()
        count = 0
        for ins in self.shader.world.instances:
            if not ins.type == 'ASSET':
                continue
            if polygon.contains(Point([ins.transform[i] for i in [3, 7, 11]])):
                count += 1
        return count >= num_furniture

    def delete_cameras_in_room(self, room):
        polygon = room.gen_polygon()
        for camera in self.shader.world.cameras:
            if polygon.contains(Point([camera.position[axis] for axis in "xyz"])):
                self.shader.world.delete_entity(camera)

    def process(self):
        # We only use rooms with more than 4 assets
        for room in self.shader.world.rooms:
            if not self.is_valid_room(room, 4):
                self.delete_cameras_in_room(room)


class CameraSetting(EntityProcessor):
    def process(self):
        for camera in self.shader.world.cameras:
            camera.set_attr("imageWidth", 1024)
            camera.set_attr("imageHeight", 512)
            camera.set_attr("cameraType", "PANORAMA")


class FurnitureLayoutSampler(SceneProcessor):
    # Randomize layout
    def process(self):
        for room in self.shader.world.rooms:
            room.randomize_layout(self.shader.world)


class EntityRandomizer(EntityProcessor):
    def randomize_model_material(self):
        for instance in self.shader.world.instances:
            # Randomize material
            self.shader.world.replace_material(
                id=instance.id,
                type='REPLACE_ALL'
            )
            if instance.type == 'ASSET':
                # Randomize model
                self.shader.world.replace_model(
                    id=instance.id
                )

    def randomize_light(self):
        for light in self.shader.world.lights:
            # Randonly adjust the color temperature
            light._tune_temp(1)
            # Randomly adjust the light intensity
            light.tune_random(1.2)

    def randomize_camera(self):
        for camera in self.shader.world.cameras:
            random_vec = np.random.normal(0, 1, size=3)
            camera_pos = np.array(list(camera.position.values()))
            randomized_pos = camera_pos + random_vec * \
                np.array([500.0, 500.0, 100.0])
            camera.set_attr(
                'position', x=randomized_pos[0], y=randomized_pos[1], z=randomized_pos[2])
            camera.set_attr('lookAt', z=randomized_pos[2])

    def process(self):
        self.randomize_model_material()
        self.randomize_light()
        self.randomize_camera()


class Render(RenderProcessor):
    def process(self, *args, **kwargs):
        self.gen_rgb(distort=0, noise=0)


class SemanticOutput(PixelProcessor):
    def modify_nyu40_labels(self):
        # modify NYU40 labels for aligning with label set of '2D-3D-S' dataset
        s_map = pd.read_csv(NYU40_MAPPING)
        s_map = s_map[["id", "nyu_label_id"]]
        s_map.index = s_map["id"]
        s_map.drop(columns="id", inplace=True)
        s_map.loc[
            ~s_map["nyu_label_id"].isin(
                [1, 2, 5, 6, 7, 8, 9, 10, 14, 15, 22]
            ),
            "nyu_label_id"
        ] = 0
        s_map.loc[s_map["nyu_label_id"] == 5, "nyu_label_id"] = 3
        s_map.loc[s_map["nyu_label_id"] == 6, "nyu_label_id"] = 4
        s_map.loc[s_map["nyu_label_id"] == 8, "nyu_label_id"] = 5
        s_map.loc[s_map["nyu_label_id"] == 9, "nyu_label_id"] = 6
        s_map.loc[(s_map["nyu_label_id"] == 7) | (s_map["nyu_label_id"] == 14), "nyu_label_id"] = 9
        s_map.loc[(s_map["nyu_label_id"] == 10) | (s_map["nyu_label_id"] == 15), "nyu_label_id"] = 7
        s_map.loc[s_map["nyu_label_id"] == 22, "nyu_label_id"] = 8
        s_map = s_map.to_dict()["nyu_label_id"]
        return s_map

    def process(self, **kwargs):
        self.gen_semantic(label_arch=self.modify_nyu40_labels())
```

Code 2: DSL code for semantic segmentation task.

```python
import math
import numpy as np
from shapely.ops import nearest_points
from shapely.geometry import Point

from ksecs.ECS.processors.pixel_processor import PixelProcessor
from ksecs.ECS.processors.render_processor import RenderProcessor
from ksecs.ECS.processors.entity_processor import EntityProcessor


class CameraSetter(EntityProcessor):
    # Set a camera in the room by a heuristic rule
    def set_camera(self, room):
        width = 640
        height = 480
        # look from right corner of the room
        camera_height = min(2000, self.shader.world.levels[0].height)
        right_corner = np.max(room.boundary, axis=0)
        room_polygon = room.gen_polygon()
        border, point = nearest_points(room_polygon, Point(right_corner))
        pos = [border.x - 200, border.y - 200, camera_height - 200]
        # look at room center
        look_at = room.position + [camera_height / 2]

        hfov = 57
        tan = (math.tan(math.radians(hfov / 2))) / (width / height)
        vfov = math.degrees(math.atan(tan)) * 2

        self.shader.world.add_camera(
            id=f"view_{room.roomId}",
            cameraType="PERSPECTIVE",
            hfov=hfov,
            vfov=vfov,
            imageWidth=width,
            imageHeight=height,
            position=pos,
            lookAt=look_at,
            up=[0, 0, 1]
        )

    def is_valid_room(self, room, num_furniture):
        # Check if the number of furniture in the room is above threshold.
        polygon = room.gen_polygon()
        count = 0
        for ins in self.shader.world.instances:
            if not ins.type == 'ASSET':
                continue
            if polygon.contains(Point([ins.transform[i] for i in [3, 7, 11]])):
                count += 1
        return count >= num_furniture

    def process(self):
        # Delete all existing cameras
        for camera in self.shader.world.cameras:
            self.shader.world.delete_entity(camera)

        # Set new camera for valid room
        for room in self.shader.world.rooms:
            if self.is_valid_room(room, 4):
                self.set_camera(room)


class Render(RenderProcessor):
    def process(self, *args, **kwargs):
        self.gen_rgb(distort=0, noise=0)


class DepthOutput(PixelProcessor):
    def process(self, **kwargs):
        self.gen_depth(distort=0, noise=0)
```

Code 3: DSL code for depth estimation task.

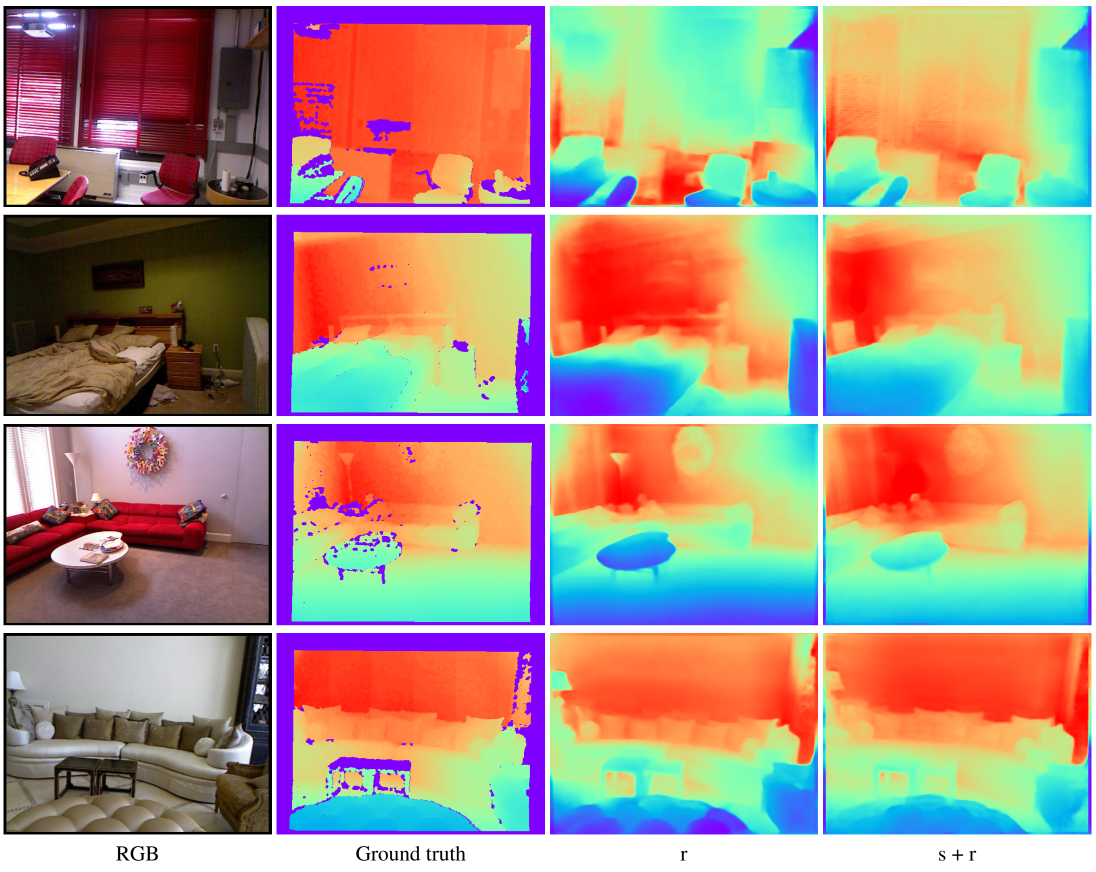


Figure 1: Qualitative results of the depth estimation with AdaBins [BAW21].

```python
class FurnitureLayoutSampler(SceneProcessor):
    def process(self):
        for room in self.shader.world.rooms:
            room.randomize_layout(self.shader.world)
```

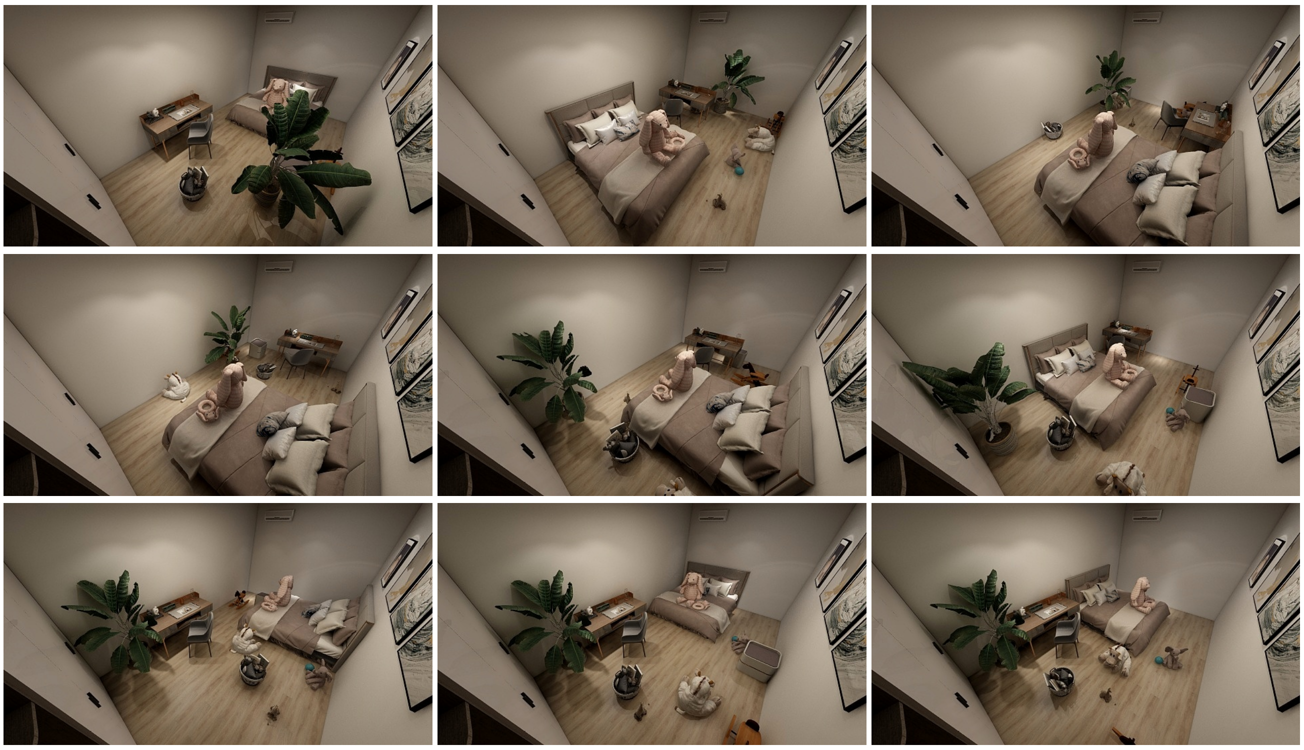


Figure 2: More results generated by the scene-level sampler.

```
class LightSampler(EntityProcessor):
    def process(self):
        for light in self.shader.world.lights:
            light.tune_temp() # randomize color temperature
            light.tune_intensity() # randomize intensity
```

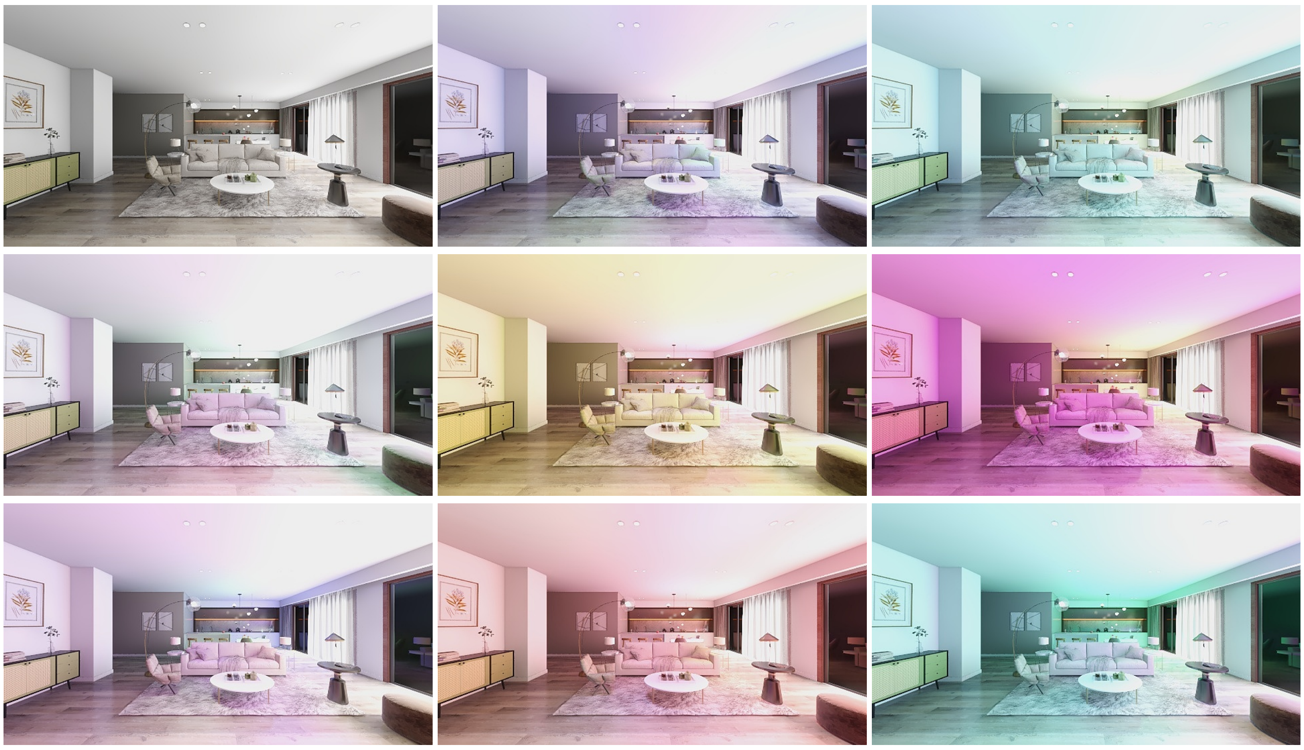


Figure 3: More results generated by the light sampler.

```python
class ModelSampler(EntityProcessor):
    def process(self):
        for instance in self.shader.world.instances:
            if instance.type == 'ASSET':
                self.shader.world.replace_model(id=instance.id)
```

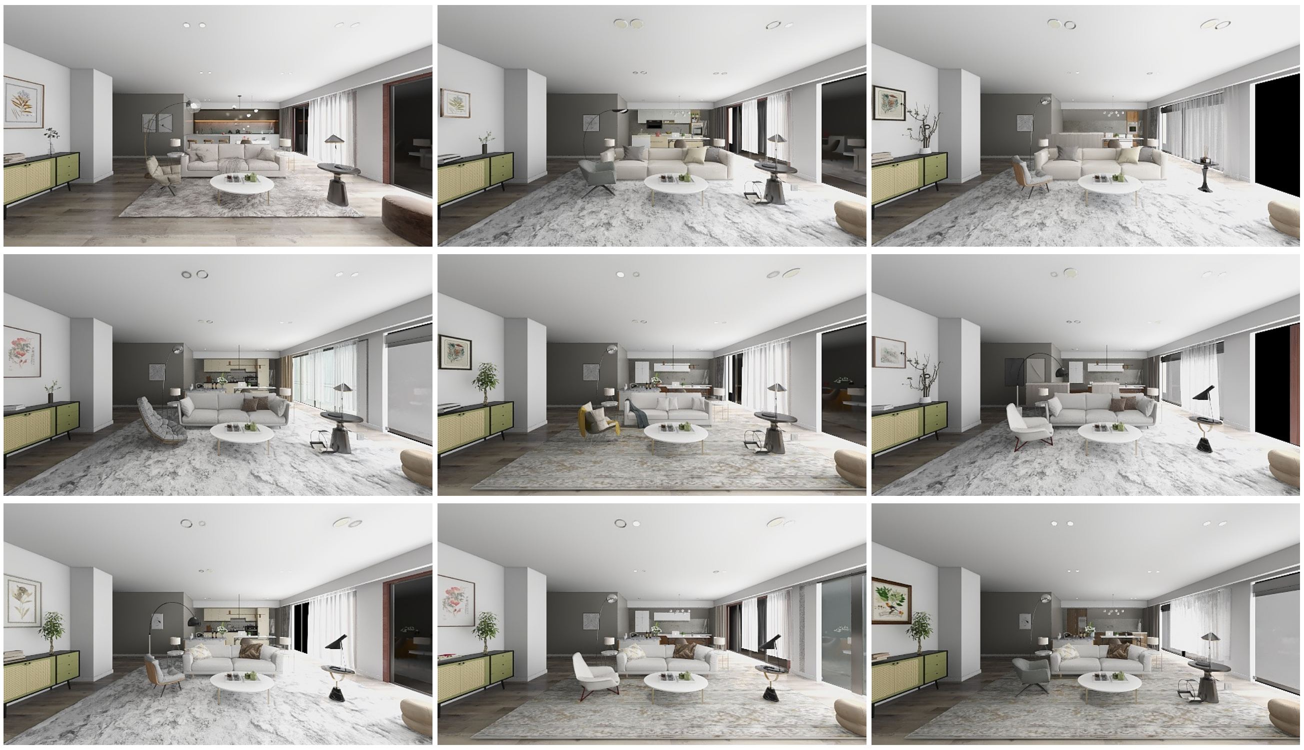


Figure 4: More results generated by the model sampler.

```python
class MaterialSampler(EntityProcessor):
    def process(self):
        for instance in self.shader.world.instances:
            # 118: 'floor'
            # 5: 'sofa'
            # 116: 'carpet'
            if instance.label in [118, 5, 116]:
                self.shader.world.replace_material(id=instance.id)
```

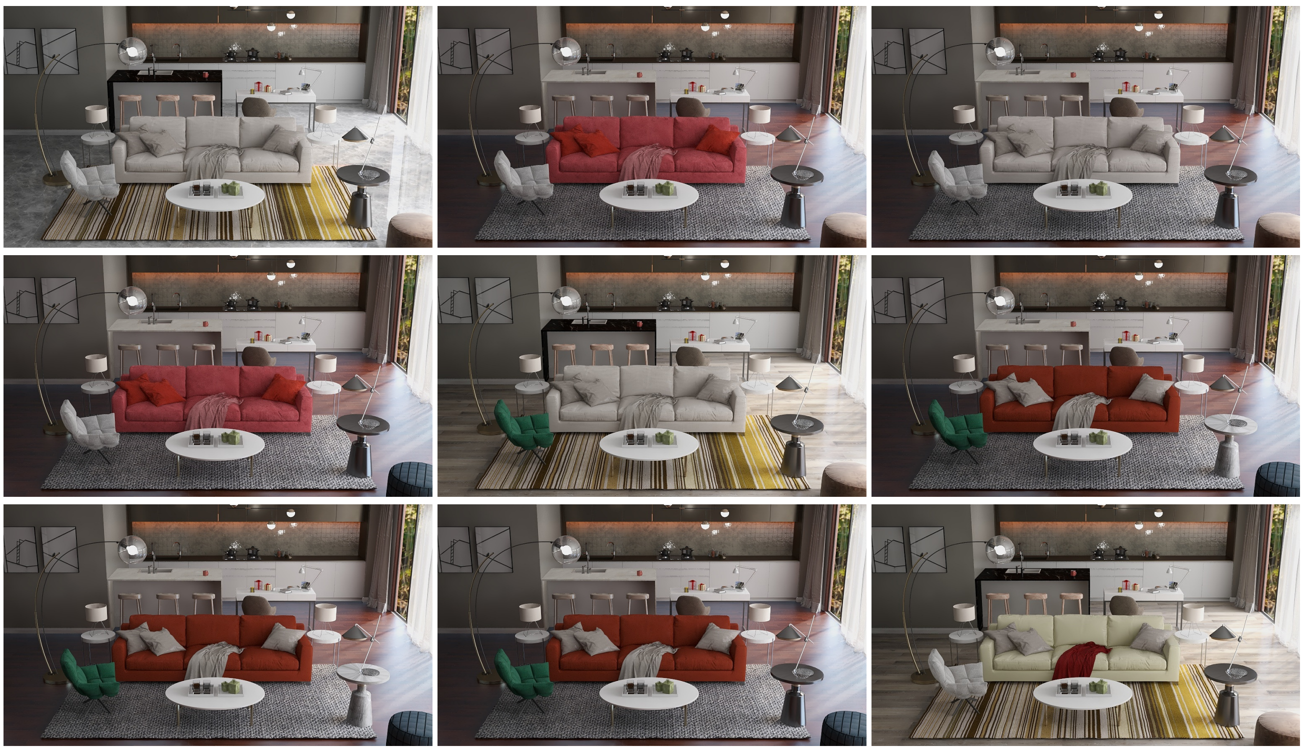
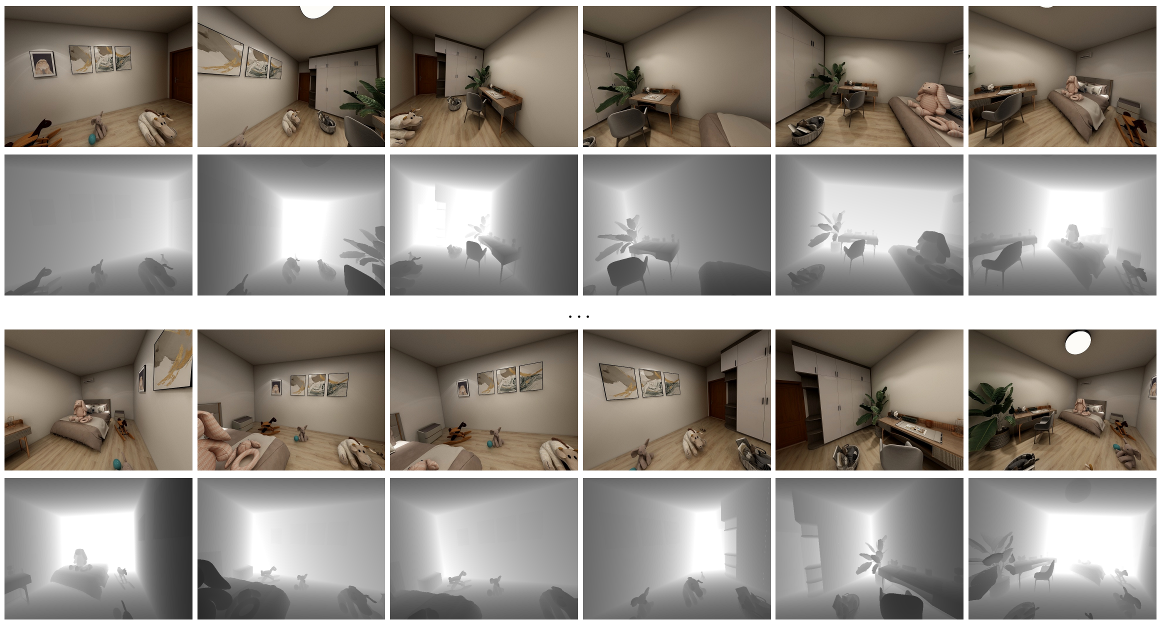


Figure 5: More results generated by the material sampler.

```python
import numpy as np
from math import sqrt
from shapely.geometry import Point
from shapely.ops import nearest_points
import glm
from ksecs.ECS.processors.entity_processor import EntityProcessor


class CustomTrajectorySampler(EntityProcessor):

    def calculate_vel(self, velocity, step_time):
        mu = glm.normalize(glm.vec2(np.random.normal(0, 1, size=2)))
        FORCE = 100
        f = FORCE * mu
        PHI, A, C_D, M = 1.204, 0.09, 0.1, 1.0
        d = -0.5 * glm.normalize(velocity) * PHI * A * C_D * glm.dot(velocity, velocity)
        velocity = velocity + (d + f) * M * step_time
        S_MAX = 10
        if S_MAX < sqrt(glm.dot(velocity, velocity)):
            velocity = S_MAX * glm.normalize(velocity)
        return velocity

    def process(self):
        for camera in self.shader.world.cameras:
            self.shader.world.delete_entity(camera)

        key_points = []
        for room in self.shader.world.rooms:
            # init
            room_points = []
            room_polygon = room.gen_polygon()
            camera_vel = glm.normalize(glm.vec2(np.random.normal(0, 1, size=2)))
            camera_pos = glm.vec2(room.position)
            # calcaulate key points
            length_of_trajectory, delta_time, scale = 5, 0.03, 1000
            for i in range(length_of_trajectory):
                # update camera params
                camera_vel = self.calculate_vel(camera_vel, delta_time)
                new_position = camera_pos + scale * camera_vel * delta_time
                next_camera_point = Point(tuple(new_position.xy))
                if not next_camera_point.within(room_polygon):
                    p1, p2 = nearest_points(room_polygon, next_camera_point)
                    normal = glm.normalize(glm.vec2(p1.x - p2.x, p1.y - p2.y))
                    camera_vel = glm.reflect(camera_vel, normal)
                camera_pos = camera_pos + scale * camera_vel * delta_time
                room_points.append(list(camera_pos))
            key_points.append(room_points)

        self.make_traj(
            imageHeight=960,
            imageWidth=1280,
            keyPoints=key_points,
            speed=1200,
            fps=3,
            speedMode=1,
            pitchMode=1,
            pitch=[-10, 10],
            hfov=70,
            vfov=55,
            height=1400,
            heightMode=1,
            cameraType="PERSPECTIVE"
        )
```

Code 4: DSL code for custom trajectory sampler.


(a)

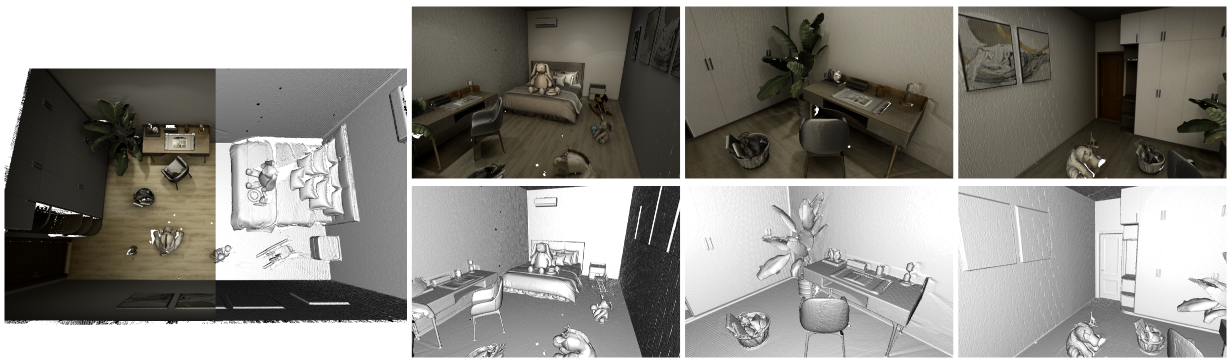

(b)

Figure 6: Results of custom trajectory sampler. (a) Sampled trajectory. (b) 3D reconstruction result by BundleFusion [DNZ*17].