MINERVAS: Massive INterior EnviRonments VirtuAl Synthesis Supplementary Material

Haocheng Ren[1,*], Hao Zhang[1,*], Jia Zheng[2], Jiaxiang Zheng[2], Rui Tang[2], Rui Wang[1], Hujun Bao[1]
[1]State Key Lab of CAD&CG, Zhejiang University [2]Manycore Tech (Kujiale)
https://coohom.github.io/MINERVAS

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

1. Experiment on Room Layout Estimation

DSL code. We first filter the room under Manhattan-world assumption using DSL in the Scene Process Stage. We set the camera model as "panorama" and the camera resolution as 1024×512 in the Entity Process Stage. We also use the sampler of the transformation component to randomly move cameras in each room, and export corner and camera positions. Figure 1 shows the DSL code for this task.

Experimental setup. In this experiment, we use MatterportLayout [2, 5] as the real dataset. The dataset consists of 1,647 images for training, 190 images for validation, and 458 images for testing. We use our system to synthesize 120K panorama images from 80K scenes. Each panorama image corresponds to one room in scenes.

Following [5], we adopt four standard metrics: 3D IoU, 2D IoU, RMSE and the accuracy under the threshold (δ_1). We adopt HorizonNet [4] as the baseline approach. We use an Adam optimizer with an initial learning rate of 3×10^{-4} with a polynomial decay policy. We set the mini-batch size to 24. We also use two training strategies in this experiment, *i.e.*, "r" and "s + r". In "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.

Qualitative results. We show more qualitative results of room layout estimation in Figure **??**.

2. Experiment on Semantic Segmentation

DSL code. We set the camera model as "panorama" and the camera resolution as 1024×512 in the Entity Process Stage, and output the customized semantic label in the Pixel Process Stage. Figure 2 shows the DSL for this task.

Experimental setup. In this experiment, we use 2D-3D-S [1] as the real data. We split the dataset into 955 for training, 84 for validation, and 373 for testing. Then, we synthesize 12k panoramic images using our system. Each panorama image corresponds to one room in scenes.

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 10k iterations.

Qualitative results. We show more qualitative results of semantic segmentation in Figure 3. As can be seen, training on the synthetic and real dataset achieves the best result. The boundary is more clear after using synthetic dataset. It demonstrates that our synthetic data could be used to improve the performance of the network.

3. Experiment on Depth Estimation

DSL code. We set the resolution of the image to 640×480 and horizontal field-of-view (FoV) to 53° in the Entity Process Stage, which are the same as Microsoft Kinect used in the NYUv2 dataset. Figure 4 shows the DSL for this task.

Experimental setup. In this experiment, we use NYUv2 as the real data. We split the images into 795 for training and 654 for testing. Then, we synthesize 14k images using our system.

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 each strategy, we train the whole network for 15k iterations.

Qualitative results. We show more qualitative results in Figure 5. As one can see, training on the synthetic and real dataset, the network generates more accurate estimations

*: Equal contribution.

than that only using real images for training. The noise has been significantly reduced in the depth estimation.

4. Additional Results of Random Samplers

Furniture Layout sampler. With the layout sampler, users can change the furniture arrangement in the scene process stage easily, which could further increase the diversity of the scene. We show the code example using the furniture layout sampler and then give more results in Figure 6.

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 7.

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 8.

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 9.

Custom sampler. The flexibility of our DSL allows users to create their customized sampling strategy. Figure 10 shows an example of customized trajectory sampler DSL. We can further use this customized sampler for vision tasks, *e.g.*, SLAM as in Figure 11.

References

[1] Iro Armeni, Sasha Sax, Amir R Zamir, and Silvio Savarese. Joint 2d-3d-semantic data for indoor scene understanding. *arXiv preprint arXiv:1702.01105*, 2017. 1

[2] Angel X. Chang, Angela Dai, Thomas A. Funkhouser, Maciej Halber, Matthias Nießner, Manolis Savva, Shuran Song, Andy Zeng, and Yinda Zhang. Matterport3d: Learning from RGB-D data in indoor environments. In *3DV*, pages 667–676, 2017. 1

[3] Angela Dai, Matthias Nießner, Michael Zollhöfer, Shahram Izadi, and Christian Theobalt. Bundlefusion: Real-time globally consistent 3d reconstruction using on-the-fly surface reintegration. *ACM Transactions on Graphics (TOG)*, 36(4):1, 2017. 9

[4] Cheng Sun, Chi-Wei Hsiao, Min Sun, and Hwann-Tzong Chen. Horizonnet: Learning room layout with 1d representation and pano stretch data augmentation. In *CVPR*, pages 1047–1056, 2019. 1

[5] Chuhang Zou, Jheng-Wei Su, Chi-Han Peng, Alex Colburn, Qi Shan, Peter Wonka, Hung-Kuo Chu, and Derek Hoiem. Manhattan room layout reconstruction from a single 360 image: A comparative study of state-of-the-art methods. *International Journal of Computer Vision (IJCV)*, 2021. 1

```python
class ManhattanSceneFilter(SceneProcessor):
    def filter_scene(self):
        Valid = False
        for room in self.shader.world.rooms:
            corners = room.boundary
            shift_corners = copy.deepcopy(corners)
            shift_corners.append(shift_corners.pop(0))
            for i in range(len(shift_corners)):
                direction = glm.vec2(shift_corners[i]).xy - glm.vec2(corners[i]).xy
                direction = glm.normalize(direction)
                shift_corners[i] = direction
            EPSILON = 0.001
            MANHATTAN = True
            for i in range(len(shift_corners)):
                cos = glm.dot(shift_corners[i], shift_corners[(i + 1) % len(shift_corners)])
                if cos > EPSILON:
                    MANHATTAN = False
                    break
            if MANHATTAN:
                Valid = True
        return Valid

    def process(self):
        if self.filter_scene():
            print("get the expected scene")
        else:
            sys.exit(7) # Don't process this scene anymore.

```

```python
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")
            camera.set_attr("position", x=1000, y=1000, z=800)
```

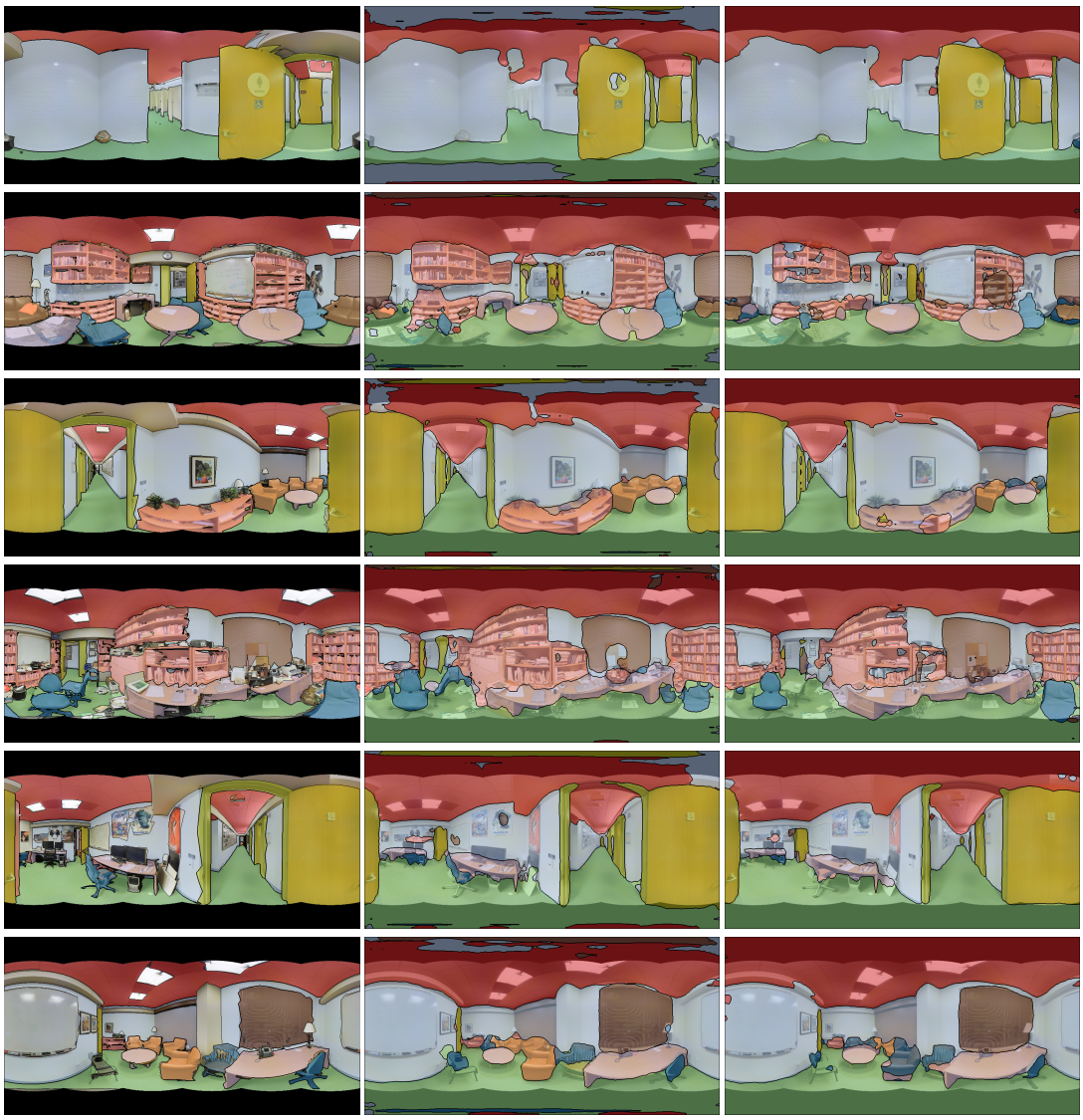
```python
class UserOutput(StructureProcessor):
    def process(self):
        # write out the corners of the rooms in the scene
        for room in self.shader.world.rooms:
            for plane, height in zip(["floor", "ceiling"], [0, self.shader.world.levels[0].height]):
                corners = []
                for corner in room.boundary:
                    corners.append({'x': corner[0], 'y': corner[1], 'z': height})
                self.shader.world.pick(
                    corners=corners,
                    catName=plane,
                    type='corners',
                    id=f"{room.roomId}_{plane}"
                )
        # write out cameras in the scene
        for camera in self.shader.world.cameras:
            self.shader.world.pick(
                type="camera",
                position=camera.position,
                id=camera.id
            )
```

Figure 1. DSL for Manhattan room layout estimation task.

```python
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")
```

```python
class LabelMapping(PixelProcessor):
    def process(self, **kwargs):
        self.gen_semantic(label_arch=NYU40_Mapping)
```

Figure 2. DSL for semantic segmentation task.

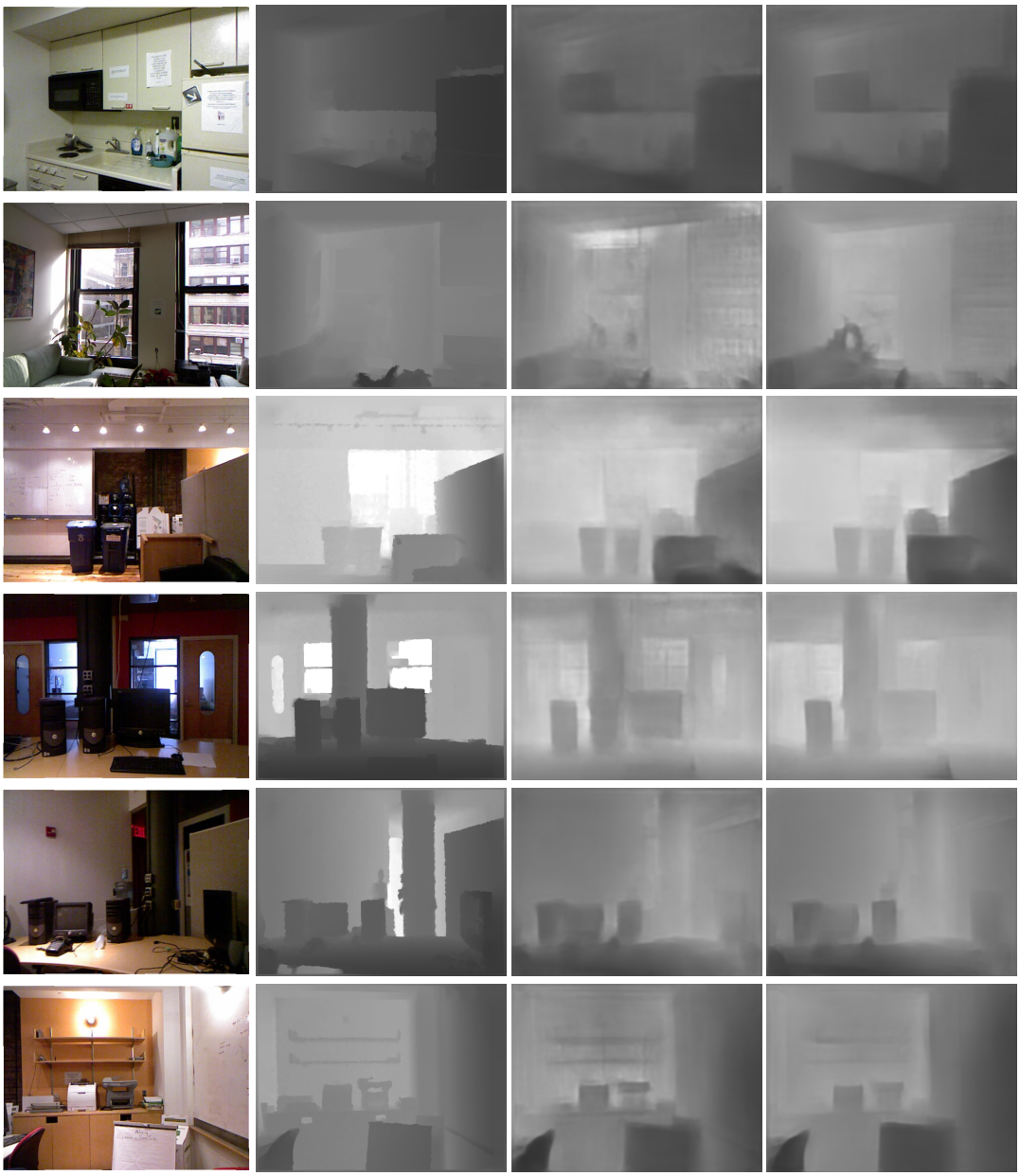


Ground truth r s + r

Figure 3. More qualitative results of the semantic segmentation on 2D-3D-S dataset. Different colors denote different semantic categories.

```
class CameraSetting(EntityProcessor):
    def process(self):
        for camera in self.shader.world.cameras:
            camera.set_attr("imageWidth", 640)
            camera.set_attr("imageHeight", 480)
            camera.set_attr("cameraType", "PERSPECTIVE")
            camera.set_attr("hfov", 53)
```

Figure 4. DSL for depth estimation.



RGB Ground truth r s + r

Figure 5. More qualitative results of the depth estimation on NYUv2 dataset.

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

Figure 6. More results generated by the scene-level sampler.

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

Figure 7. More results generated by the light sampler.

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

Figure 8. More results generated by the mesh sampler.

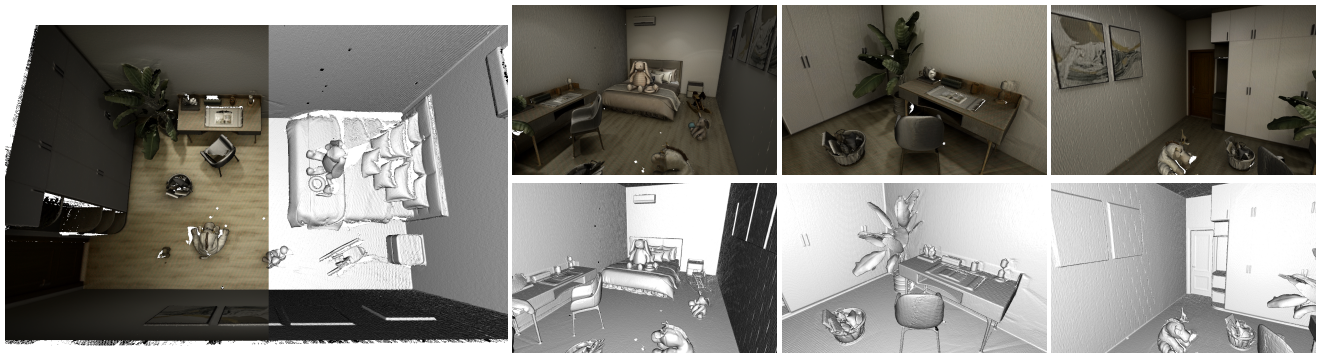
```python
class MaterialSampler(EntityProcessor):
    def process(self):
        for instance in self.shader.world.instances:
            if instance.label_name in ['sofa', 'floor', 'carpet']:
                self.shader.world.replace_material(id=instance.id)
```

Figure 9. More results generated by the material sampler.

```
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 road points
            length_of_trajectory, delta_time, scale = 100, 0.03, 1000
            for i in range(length_of_trajectory):
                # Camera
                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=480,
            imageWidth=640,
            keyPoints=key_points,
            speed=1200,
            fps=3,
            speedMode=1,
            pitchMode=1,
            pitch=[-10, 10],
            hfov=70,
            vfov=55,
            height=1400,
            heightMode=1,
            cameraType="PERSPECTIVE"
        )
```

Figure 10. DSL for custom trajectory sampler.

(a)



(b)

Figure 11. Results of custom trajectory sampler. (a) Sampled trajectory. (b) 3D reconstruction result by BundleFusion [3].