PrITTI: Primitive-based Generation of Controllable and Editable 3D Semantic Urban Scenes

Supplementary Material

In this supplementary material, we provide additional details on dataset preprocessing, implementation, and experimental setup. We also present further quantitative results evaluating generative performance and additional qualitative examples for all applications discussed in the main paper. Our supplementary video offers a brief motivation and overview of our method as well as extended visual results.

A. Dataset Details

In this section, we describe our dataset construction and the key procedures used to analyze it.

Dataset Construction. KITTI-360 [14] defines 37 labels, which we merge into 16 semantic categories, covering all major urban scene components. As an initial step, we analyze the dataset statistics and remove a few extremely rare labels including tunnel, bridge, train, stop, and caravan. Among the remaining ones, five correspond to ground-related classes (road, terrain, sidewalk, parking, and ground), which we treat as distinct ground categories. The rest are grouped into 11 object categories based on both semantic and physical scale similarity, the latter being important for accurate primitive reconstruction. A complete mapping of semantic classes to object categories is provided in Tab. 1. For detailed definitions of individual classes, we refer readers to [14].

For each ego-vehicle pose, we adopt its IMU coordinate system as the local reference frame and define a $64\,m \times 64\,m$ 3D layout centered on the vehicle. The field of view (FOV) extends $64\,m$ in the forward direction and $32\,m$ laterally on each side. To avoid including distant or marginally visible objects, we retain only primitives whose 3D centers lie within this FOV and ground polygons with at least one vertex contained in it. Poses without any primitive annotations (typically at sequence boundaries) are excluded. The provided KITTI-360 test poses are sequentially distributed over limited areas of each sequence, covering only a narrow spatial extent. In short sequences, they typically span one continuous region, while in longer sequences, they may occupy two disjoint areas. Consequently, the original split insufficiently represents the full spatial diversity of the scenes.

To address this, we construct a custom train-test split. We first compute the total driving distance across all KITTI-360 sequences and divide the full trajectory into 100 equal-distance segments. Within each segment, the majority of poses are assigned to the training set, while a $15\,m$ region near the end is reserved for testing. To minimize spatial overlap with training poses, we exclude $5\,m$ margins on either side of the test regions. Furthermore, we remove any test pose located within $5\,m$ in the Bird's Eye View (BEV) of a training pose, and vice versa. This prevents overlap caused by the ego-vehicle revisiting the same location from opposite directions. The final split comprises 61,913 training poses and 1,233 test poses, ensuring broad and non-overlapping spatial coverage across the entire dataset.

Primitive Count Selection. We adopt a fixed-size set prediction strategy for object primitives rather than an autoregressive decoding scheme, as this offers clear advantages for large-scale 3D scenes. Predicting all primitives in parallel enables the model to handle hundreds of objects in a single forward pass, whereas an autoregressive decoder would require sequential sampling, leading to higher inference time and cumulative prediction errors. In addition, generating the entire scene concurrently avoids imposing an arbitrary ordering on objects, which is inconsistent with the inherently unordered nature of scene layouts. Accordingly, we design the object decoder $\mathcal{D}_{\mathcal{O}}$ (see Sec. B.2) to infer a fixed-size set of N^c primitives for each category c. We determine N^c by analyzing the distribution of ground-truth primitive counts and selecting the 95$^{\text{th}}$ percentile as the cutoff. The resulting values for each category are listed in Tab. 1. During preprocessing, if the count of instances in a category exceeds its predefined threshold in a given scene, we retain only the N^c closest objects to the ego-vehicle and discard the farthest ones. This ensures that all scenes, regardless of their density, are processed. Importantly, not all decoded objects are necessarily retained in the final scene. Each predicted primitive is associated with an existence probability, and only those exceeding a predefined threshold are kept. This probabilistic filtering allows the model to naturally handle both sparse and dense scenes while maintaining a fixed-size set.

Scene Labeling. To train our class-conditional latent diffusion model (see Sec. B.3), each scene is assigned a discrete label reflecting its vegetation density. Since such annotations are not provided in the KITTI-360 dataset, we derive them automatically from dataset statistics. Specifically, for each scene, we compute the total number and volume of vegetation primitives, considering both cuboids and ellipsoids. A scene is then labeled as *low* if both quantities fall below their respective 25$^{\text{th}}$ percentile thresholds, *high* if both exceed the 75$^{\text{th}}$ percentile, and *medium* otherwise.

B. Methodological Details

This section describes the representations, architectural components, and implementation details of our method.

Object Category	Semantic Classes	Primitive Type	Predicted Count
vegetation cuboid ($\mathcal{VC}$)	vegetation	3D cuboid	178
vegetation ellipsoid ($\mathcal{VE}$)	vegetation	3D ellipsoid	159
vehicle big ($\mathcal{VB}$)	truck, bus	3D cuboid	2
vehicle small ($\mathcal{VS}$)	car, trailer, unknown vehicle	3D cuboid	18
two-wheelers ($\mathcal{TW}$)	motorcycle, bicycle	3D cuboid	6
human ($\mathcal{H}$)	rider, pedestrian	3D cuboid	5
construction big ($\mathcal{CB}$)	building, garage, unknown construction	3D cuboid	16
construction small ($\mathcal{CS}$)	wall, fence, guardrail, gate	3D cuboid	77
pole ($\mathcal{P}$)	small pole, big pole	3D cuboid	19
traffic control ($\mathcal{TC}$)	traffic light, traffic sign, lamp	3D cuboid	17
object ($\mathcal{O}$)	trashbin, box, unknown object	3D cuboid	17

Table 1. List of object categories defined in our method along with their corresponding semantic classes and annotated primitive types. The last column indicates the fixed number of primitives predicted per category, determined based on dataset statistics.

B.1. Cholesky Parameters

Standard quaternion-based encodings of 3D orientation suffer from sign ambiguity: both $\mathbf{q}$ and $-\mathbf{q}$ represent the same rotation but differ numerically. This causes discontinuities in the loss landscape when optimized with standard regression objectives. Similarly, eigendecomposition-based encodings are ambiguous because eigenvectors are only defined up to sign (i.e. both $\mathbf{v}$ and $-\mathbf{v}$ are valid but numerically distinct), leading to comparable instability during gradient-based training. To overcome these issues, we propose a Cholesky-based parameterization that jointly encodes orientation and size in a continuous 6D representation.

Scatter Matrix Construction. To derive this representation, we first define a scatter matrix $\mathbf{S} \in \mathbb{R}^{3\times3}$ that describes the spatial extent of an object in 3D space. Let $\lambda_1, \lambda_2, \lambda_3 > 0$ denote the object's scaling factors along its principal axes, and let $\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3 \in \mathbb{R}^3$ be the orthonormal column vectors of its rotation matrix. Defining the orthonormal basis matrix $\mathbf{V} = [\mathbf{v}_1\ \mathbf{v}_2\ \mathbf{v}_3]$ and the diagonal scale matrix $\mathbf{\Lambda} = \mathrm{diag}(\lambda_1, \lambda_2, \lambda_3)$, the scatter matrix is given by:

$$\mathbf{S} = \mathbf{V}\mathbf{\Lambda}\mathbf{V}^\top = \sum_{j=1}^{3} \lambda_j \mathbf{v}_j \mathbf{v}_j^\top \tag{1}$$

Cholesky Decomposition. We apply the Cholesky decomposition [6] to the scatter matrix $\mathbf{S}$, yielding

$$\mathbf{S} = \mathbf{L}\mathbf{L}^\top \tag{2}$$

where $\mathbf{L} \in \mathbb{R}^{3\times3}$ is a lower-triangular matrix. The Cholesky parameters $\mathbf{c} \in \mathbb{R}^6$ are defined as the six non-zero entries of $\mathbf{L}$. This representation removes orientation ambiguities by operating on the sign-invariant scatter matrix $\mathbf{S}$: since each outer product $\mathbf{v}_j\mathbf{v}_j^\top$ is unchanged under sign flips $\mathbf{v}_j \mapsto -\mathbf{v}_j$, $\mathbf{S}$ remains invariant.

Scatter Matrix Properties. Additionally, we can show that $\mathbf{S}$ is symmetric and positive-definite:
Symmetric:

$$\mathbf{S}^\top = (\mathbf{V}\mathbf{\Lambda}\mathbf{V}^\top)^\top = (\mathbf{V}^\top)^\top\mathbf{\Lambda}^\top\mathbf{V}^\top = \mathbf{V}\mathbf{\Lambda}\mathbf{V}^\top = \mathbf{S} \tag{3}$$

since $\mathbf{\Lambda}$ is diagonal and therefore symmetric.
Positive-definite: Because $\mathbf{V}$ is orthonormal (i.e. $\mathbf{V}^\top = \mathbf{V}^{-1}$), $\mathbf{S}$ is similar to $\mathbf{\Lambda}$:

$$\mathbf{S} = \mathbf{V}\mathbf{\Lambda}\mathbf{V}^{-1} \tag{4}$$

and thus shares the same eigenvalues $\lambda_1, \lambda_2, \lambda_3 > 0$.
These properties guarantee a unique Cholesky decomposition [6] with a lower-triangular matrix $\mathbf{L}$ having strictly positive diagonal entries. This in turn defines a *unique* 6D representation $\mathbf{c} \in \mathbb{R}^6$, well-suited for regression-based learning.
Reconstruction. Given predicted Cholesky parameters $\hat{\mathbf{c}}$, we reshape them into a lower-triangular matrix $\hat{\mathbf{L}} \in \mathbb{R}^{3\times3}$ and recover the scatter matrix as $\hat{\mathbf{S}} = \hat{\mathbf{L}}\hat{\mathbf{L}}^\top$. Performing eigendecomposition of $\hat{\mathbf{S}}$ yields its eigenvalues $\hat{\lambda}_1, \hat{\lambda}_2, \hat{\lambda}_3$ and eigenvectors $\hat{\mathbf{v}}_1, \hat{\mathbf{v}}_2, \hat{\mathbf{v}}_3$. These define the object's scale matrix $\hat{\mathbf{\Lambda}} = \mathrm{diag}(\hat{\lambda}_1, \hat{\lambda}_2, \hat{\lambda}_3)$ and rotation matrix $\hat{\mathbf{R}} = [\hat{\mathbf{v}}_1\ \hat{\mathbf{v}}_2\ \hat{\mathbf{v}}_3]$.

B.2. Layout Variational Autoencoder

In the first training stage, a Layout Variational Autoencoder (LVAE) learns to compress 3D semantic scene layouts into compact 2D latent representations suitable for subsequent latent diffusion modeling.

B.2.1. Architecture

The LVAE comprises separate encoder-decoder pairs for the ground and object modalities.
Ground Encoder. Our ground encoder $\mathcal{E}_\mathcal{G}$ consists of stacked downsampling blocks that progressively reduce spatial resolution while increasing channel dimensionality. Each

block contains a residual unit followed by a strided convolution. Residual units include two convolutional layers with group normalization and SiLU activation. At the bottleneck, a self-attention layer captures global context, while additional residual blocks refine local features. The encoder takes as input the ground raster maps, $[\mathbf{H}; \mathbf{B}] \in \mathbb{R}^{256\times256\times10}$, and outputs a latent grid of shape $32 \times 32 \times 64$. This grid is split along the channel dimension into two chunks representing the mean $\boldsymbol{\mu}_{\mathcal{G}} \in \mathbb{R}^{32\times32\times32}$ and variance $\boldsymbol{\sigma}_{\mathcal{G}} \in \mathbb{R}^{32\times32\times32}$ of the ground latent representation.

Object Encoder. The object encoder $\mathcal{E}_{\mathcal{O}}$ operates on a fixed-size set of object primitives, each represented by a 20D feature vector. These vectors are first projected to a 512D embedding space and then processed by a 6-layer Transformer encoder with 8 attention heads and a feedforward dimension of 1024. A subsequent linear projection layer reduces the output dimensionality to 64. The resulting per-object features are spatially arranged into a latent grid of shape $32 \times 32 \times 64$ using a scatter-mean operation. Similar to the ground branch, the grid is split along the channel dimension into the mean $\boldsymbol{\mu}_{\mathcal{O}} \in \mathbb{R}^{32\times32\times32}$ and variance $\boldsymbol{\sigma}_{\mathcal{O}} \in \mathbb{R}^{32\times32\times32}$ of the object latent representation.

Joint Layout Latent. We construct joint mean and variance tensors, $\boldsymbol{\mu}_{\mathcal{L}}, \boldsymbol{\sigma}_{\mathcal{L}} \in \mathbb{R}^{32\times32\times64}$, by concatenating the corresponding ground and object components along the channel axis. A joint layout latent representation $\mathbf{z}_{\mathcal{L}} \in \mathbb{R}^{32\times32\times64}$ is then sampled via the reparameterization trick. This formulation maintains a disentangled latent structure, enabling straightforward separation into ground and object latents, $\mathbf{z}_{\mathcal{G}}, \mathbf{z}_{\mathcal{O}} \in \mathbb{R}^{32\times32\times32}$. We found this design beneficial for improving reconstruction fidelity and enabling further applications such as ground-conditioned primitive generation.

Ground Decoder. The ground decoder $\mathcal{D}_{\mathcal{G}}$ processes the ground latent representation $\mathbf{z}_{\mathcal{G}}$ through a self-attention layer followed by two residual units. The resulting features are progressively upsampled to the original spatial resolution of 256×256 using a cascade of upsampling blocks, each comprising two residual units and a transposed convolution. As the spatial resolution increases, the channel dimensionality is gradually reduced. A final convolutional layer maps the output to 10 channels, matching the dimensionality of the input ground raster maps, $[\hat{\mathbf{H}}; \hat{\mathbf{B}}] \in \mathbb{R}^{256\times256\times10}$.

Object Decoder. The object decoder $\mathcal{D}_{\mathcal{O}}$ adopts a DETR [5] Transformer architecture with N learnable queries representing the total number of predicted primitives across all object categories. It consists of 6 Transformer decoder layers with 8 attention heads, a feedforward dimension of 1024, and no dropout. The queries interact with the object latent $\mathbf{z}_{\mathcal{O}}$ through cross-attention. Specifically, $\mathbf{z}_{\mathcal{O}}$ is first patchified with a patch size of 1, producing a sequence of 32×32 tokens augmented with sine positional embeddings. The resulting embeddings, each of dimension 512, are grouped by category and passed to category-specific prediction heads.

Prediction Heads. Each object category employs a dedicated 3-layer feedforward network with a hidden size of 1024. For every query, the head outputs a 9D vector encoding the normalized 3D center location and 6D Cholesky parameters of its corresponding primitive. An additional linear layer predicts its probability of existence.

B.2.2. Training

The first-stage training loss consists of three components, described below.

Ground Loss. We supervise the reconstruction of input raster maps using: (i) a binary cross-entropy (BCE) loss for occupancy masks and (ii) an L1 loss for height maps, averaged over occupied pixels. Let $\mathbf{H} \in \mathbb{R}^{H\times W\times5}$ and $\mathbf{B} \in \{0,1\}^{H\times W\times5}$ denote the ground-truth height maps and occupancy masks, and $\hat{\mathbf{H}}$ and $\hat{\mathbf{B}}$ their predicted counterparts. The ground loss is defined as:

$$\mathcal{L}_{\text{ground}} = \lambda_{\mathcal{G}}^{\text{occ}} \cdot \mathcal{L}_{\text{BCE}}(\hat{\mathbf{B}}, \mathbf{B}) + \lambda_{\mathcal{G}}^{\text{height}} \cdot \tfrac{1}{\sum \mathbf{B}} \left\| (\hat{\mathbf{H}} - \mathbf{H}) \odot \mathbf{B} \right\|_1 \quad (5)$$

where $\odot$ denotes element-wise multiplication. We set the loss weights $\lambda_{\mathcal{G}}^{\text{occ.}} = 1$ and $\lambda_{\mathcal{G}}^{\text{height}} = 9$ to balance the relative contributions of the two terms.

Object Loss. The object decoder $\mathcal{D}_{\mathcal{O}}$ predicts a fixed number of N^c primitives per object category c (see Tab. 1), leading to $N = \sum_{c=1}^{11} N^c$ total predictions per scene. Because the outputs are unordered, we establish one-to-one correspondences between predicted and ground-truth primitives using bipartite matching for each category independently. Specifically, let $\mathcal{O}^c = \{\mathbf{o}_i\}_{i=1}^{N^c}$ and $\hat{\mathcal{O}}^c = \{\hat{\mathbf{o}}_j\}_{j=1}^{N^c}$ denote the sets of ground-truth and predicted primitives for category c, respectively. Each primitive is parameterized as $\mathbf{o}_i = (\mathbf{t}_i, \mathbf{c}_i, p_i)$, where $\mathbf{t}_i \in \mathbb{R}^3$ is its 3D center location, $\mathbf{c}_i \in \mathbb{R}^6$ are the Cholesky parameters, and $p_i \in \{0,1\}$ distinguishes valid primitives ($p_i = 1$) from padded placeholders ($p_i = 0$). We apply the Hungarian algorithm [11] to find the optimal assignment $\hat{\sigma}^c$ minimizing the total matching cost:

$$\hat{\sigma}^c = \arg\min_{\sigma^c \in \mathcal{P}_{N^c}} \sum_{i=1}^{N^c} \mathcal{L}_{\text{match}}(\mathbf{o}_i, \hat{\mathbf{o}}_{\sigma(i)}) \quad (6)$$

where $\mathcal{P}_{N^c}$ is the set of all permutations of N^c elements. The pairwise matching cost is defined as:

$$\begin{aligned} \mathcal{L}_{\text{match}}(\mathbf{o}_i, \hat{\mathbf{o}}_{\sigma(i)}) = {} & \lambda_{\mathcal{O}}^{\text{prob}} \mathcal{L}_{\text{BCE}}(p_i, \hat{p}_{\sigma(i)}) \\ & + \mathbb{1}_{[p_i>0]} \lambda_{\mathcal{O}}^{\text{center}} \|\mathbf{t}_i - \hat{\mathbf{t}}_{\sigma(i)}\|_1 \\ & + \mathbb{1}_{[p_i>0]} \lambda_{\mathcal{O}}^{\text{Chol}} \|\mathbf{c}_i - \hat{\mathbf{c}}_{\sigma(i)}\|_1 \end{aligned} \quad (7)$$

After obtaining the optimal assignments, we compute the object loss over all categories as:

$$\mathcal{L}_{\text{object}} = \sum_{c\in\mathcal{C}} \left(\sum_{i=1}^{N^c} \mathcal{L}_{\text{match}}(\mathbf{o}_i, \hat{\mathbf{o}}_{\hat{\sigma}^c(i)}) \right) \quad (8)$$

where $\mathcal{C}$ denotes the set of object categories. Each term in Eq. (8) is normalized by the number of real (i.e. non-padded) ground-truth primitives in the corresponding sample, and the normalized losses are averaged across the batch. The weights $\lambda_{\mathcal{O}}^{\text{prob.}}$, $\lambda_{\mathcal{O}}^{\text{center}}$, and $\lambda_{\mathcal{O}}^{\text{Chol.}}$ are configured differently for bipartite matching and for the final loss computation. For **matching**, we use category-agnostic constants: $\lambda_{\mathcal{O}}^{\text{prob.}} = 6$, $\lambda_{\mathcal{O}}^{\text{center}} = 3$, and $\lambda_{\mathcal{O}}^{\text{Chol.}} = 3$. This weighting emphasizes the BCE term to discourage pairing low-confidence predictions with true instances, while assigning equal importance to the center and Cholesky components for accurate geometric alignment. During **loss computation**, the weights vary across object categories according to their fixed number of predicted primitives. In particular, categories are grouped into low-, medium-, and high-count ranges, with higher weights assigned to higher-count groups. This adjustment compensates for normalization by the number of real primitives. Without it, high-count categories, typically containing more real instances per sample, would contribute less to the overall loss, leading to underfitting. Within each group, the center and Cholesky weights are balanced so that their corresponding losses have comparable magnitudes, ensuring that both geometric components contribute equally. The BCE term receives a smaller weight than the geometric terms, as existence confidence is already reinforced by the previous matching stage. Increasing its weight would disproportionately emphasize classification over geometry. The final group-specific weights are summarized in Tab. 2.

Group	Object Categories	$\lambda_{\mathcal{O}}^{\text{prob.}}$	$\lambda_{\mathcal{O}}^{\text{center}}$	$\lambda_{\mathcal{O}}^{\text{Chol.}}$
Low	$\mathcal{VB}, \mathcal{TW}, \mathcal{H}$	1	3	2
Medium	$\mathcal{VS}, \mathcal{CB}, \mathcal{P}, \mathcal{TC}, \mathcal{O}$	3	9	7
High	$\mathcal{VC}, \mathcal{VE}, \mathcal{CS}$	5	15	12

Table 2. Category-specific loss weights grouped by object count.

KL Loss. Following a standard VAE formulation [10], we apply a Kullback-Leibler (KL) divergence regularization on the joint layout latent. The loss is defined as:

$$\mathcal{L}_{\text{KL}} = \lambda^{\text{KL}} \cdot \left(-\tfrac{1}{2} \sum_{d=1}^{64} \left(1 + \log(\sigma_{\mathcal{L},d}^2) - \mu_{\mathcal{L},d}^2 - \sigma_{\mathcal{L},d}^2\right)\right) \quad (9)$$

where $\mu_{\mathcal{L},d}$ and $\sigma_{\mathcal{L},d}$ denote the mean and standard deviation of the d-th latent dimension. We set $\lambda^{\text{KL}} = 1\text{e}^{-6}$.

Total Loss. The overall training objective for the first stage combines the three components as:

$$\mathcal{L}_{\text{LVAE}} = \mathcal{L}_{\text{ground}} + \mathcal{L}_{\text{object}} + \mathcal{L}_{\text{KL}} \quad (10)$$

B.2.3. Layout Reconstruction

To reconstruct a 3D scene layout, we follow separate procedures for ground surfaces and objects, reflecting their distinct representations. For **ground surfaces**, we first fuse the per-class height maps into a composite height field by selecting, at each pixel, the highest elevation and recording the class that contributed it. Using a fixed label-to-color mapping, this process simultaneously yields a composite color map aligned with this height field. The resulting raster representation is then converted into a triangle mesh (see Fig. 1). Each occupied pixel (u, v) is lifted to a 3D vertex (x, y, z) using its height value and the vertex color is assigned from the composite color map at the same location. To obtain a continuous surface, we triangulate each 2×2 pixel block at indices $(i, j), (i, j+1), (i+1, j)$, and $(i+1, j+1)$ using a consistent diagonal, forming two adjacent triangles per quad. For **objects**, we first derive the rotation matrix $\hat{\mathbf{R}}$ and scale matrix $\hat{\mathbf{\Lambda}}$ from the predicted Cholesky parameters $\hat{\mathbf{c}} \in \mathbb{R}^6$, as described in Sec. B.1. We then construct the full transformation matrix $\hat{\mathbf{T}} \in \mathbb{R}^{4\times4}$ by combining these with the predicted 3D center location $\hat{\mathbf{t}} \in \mathbb{R}^3$:

$$\hat{\mathbf{T}} = \begin{bmatrix} \hat{\mathbf{R}}\hat{\mathbf{\Lambda}} & \hat{\mathbf{t}} \\ \mathbf{0}^\top & 1 \end{bmatrix} \quad (11)$$

This transformation is applied to a unit-sized, zero-centered 3D cuboid or ellipsoid (depending on the object category) to position the primitive within the reconstructed scene layout.

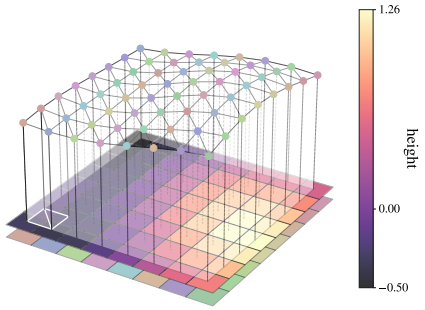


Figure 1. **Extrusion Process.** An illustrative extrusion example showing the composite color map (bottom), height field (middle), and lifted triangle mesh (top).

B.2.4. Implementation Details

The LVAE comprises 93.1M parameters and is trained using PyTorch Lightning[1] for 3 days on a single A100 GPU with a batch size of 32. Optimization is performed using AdamW with a weight decay of $5e^{-4}$, combined with a OneCycleLR scheduler. The learning rate follows a cosine annealing schedule, starting at $1e^{-4}$ and decaying to $1e^{-5}$, without a warm-up phase. During inference, we discard object primitives whose predicted existence probability falls below a threshold of $\hat{p} = 0.3$. We use a single global threshold across all semantic categories. This value controls the

[1] https://lightning.ai/docs/pytorch/stable/

trade-off between retaining more primitives and suppressing low-confident ones: lower values lead to denser but potentially noisier layouts, while higher values yield cleaner but sparser reconstructions. In practice, we found that reconstruction quality is not highly sensitive to this parameter, and values within $[0.25, 0.75]$ produce visually similar results.

B.3. Diffusion Transformer

In the second stage of our method, we train a latent diffusion model (LDM) [21] on the joint layout latent space of our pre-trained LVAE.

B.3.1. Architecture

We adopt the standard DiT architectures from [19] with a patch size of 2, yielding 16×16 tokens from the joint layout latent representation $\mathbf{z}_{\mathcal{L}} \in \mathbb{R}^{32\times32\times64}$. Tab. 3 summarizes the configurations of the DiT Base (B), Large (L), and XLarge (XL) variants used in our experiments.

Model	Layers	Hidden Size	Heads	Batch Size	Params (M)
DiT-B	12	768	12	256	130
DiT-L	24	1024	16	256	458
DiT-XL	28	1152	16	256	675

Table 3. Details of DiT-B, -L, and -XL model variants.

B.3.2. Training

Given an initial latent $\mathbf{z}_{\mathcal{L}}^0$, we define a *forward diffusion process* that gradually corrupts it by adding Gaussian noise over time, yielding a sequence of increasingly noisy latents $\mathbf{z}_{\mathcal{L}}^1, \ldots, \mathbf{z}_{\mathcal{L}}^T$. The noise magnitude at each step is controlled by a variance schedule $\{\beta_t \in (0,1)\}_{t=1}^T$. The joint distribution over all timesteps is expressed as:

$$q(\mathbf{z}_{\mathcal{L}}^{1:T} \mid \mathbf{z}_{\mathcal{L}}^0) = \prod_{t=1}^{T} q(\mathbf{z}_{\mathcal{L}}^t \mid \mathbf{z}_{\mathcal{L}}^{t-1}) \tag{12}$$

$$q(\mathbf{z}_{\mathcal{L}}^t \mid \mathbf{z}_{\mathcal{L}}^{t-1}) = \mathcal{N}(\mathbf{z}_{\mathcal{L}}^t; \sqrt{1-\beta_t}\mathbf{z}_{\mathcal{L}}^{t-1}, \beta_t\mathbf{I}) \tag{13}$$

A convenient property of the above process is that the noisy latent $\mathbf{z}_{\mathcal{L}}^t$ at any timestep t can be sampled in a single step from the following distribution:

$$q(\mathbf{z}_{\mathcal{L}}^t \mid \mathbf{z}_{\mathcal{L}}^0) = \mathcal{N}(\mathbf{z}_{\mathcal{L}}^t; \sqrt{\bar{\alpha}_t}\mathbf{z}_{\mathcal{L}}^0, (1-\bar{\alpha}_t)\mathbf{I}) \tag{14}$$

where $\alpha_t = 1 - \beta_t$, and $\bar{\alpha}_t = \prod_{s=1}^{t} \alpha_s$.

To synthesize new latents, we learn a *reverse process* that denoises from pure Gaussian noise $\mathbf{z}_{\mathcal{L}}^T \sim \mathcal{N}(0, \mathbf{I})$ back to a clean latent $\mathbf{z}_{\mathcal{L}}^0$. The joint distribution of this process is:

$$p_\theta(\mathbf{z}_{\mathcal{L}}^{0:T}) = p(\mathbf{z}_{\mathcal{L}}^T) \prod_{t=1}^{T} p_\theta(\mathbf{z}_{\mathcal{L}}^{t-1} \mid \mathbf{z}_{\mathcal{L}}^t) \tag{15}$$

$$p_\theta(\mathbf{z}_{\mathcal{L}}^{t-1} \mid \mathbf{z}_{\mathcal{L}}^t) = \mathcal{N}(\mathbf{z}_{\mathcal{L}}^{t-1}; \mu_\theta(\mathbf{z}_{\mathcal{L}}^t, t), \Sigma_\theta(\mathbf{z}_{\mathcal{L}}^t, t)) \tag{16}$$

where each transition is modeled as a Gaussian with learnable mean μ_θ and covariance Σ_θ, estimated by a denoising network parameterized by θ. In practice, we follow [9] and set $\Sigma_\theta(\mathbf{z}_{\mathcal{L}}^t, t) = \sigma_t^2\mathbf{I} = \frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t}\beta_t\mathbf{I}$, treating it as an untrained, time-dependent constant. Moreover, we represent the mean $\mu_\theta(\mathbf{z}_{\mathcal{L}}^t, t)$ as:

$$\mu_\theta(\mathbf{z}_{\mathcal{L}}^t, t) = \frac{1}{\sqrt{\alpha_t}}\left(\mathbf{z}_{\mathcal{L}}^t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\,\epsilon_\theta(\mathbf{z}_{\mathcal{L}}^t, t)\right) \tag{17}$$

where ϵ_θ predicts the noise term $\boldsymbol{\epsilon}$ in the reparameterization $\mathbf{z}_{\mathcal{L}}^t = \sqrt{\bar{\alpha}_t}\,\mathbf{z}_{\mathcal{L}}^0 + \sqrt{1-\bar{\alpha}_t}\,\boldsymbol{\epsilon}$.

For *training*, we adopt the simplified denoising objective of [9]:

$$\mathcal{L}_{\text{simple}} = \mathbb{E}_{\mathbf{z}_{\mathcal{L}}^0, \boldsymbol{\epsilon}, t}\left[\left\|\boldsymbol{\epsilon} - \epsilon_\theta\left(\sqrt{\bar{\alpha}_t}\,\mathbf{z}_{\mathcal{L}}^0 + \sqrt{1-\bar{\alpha}_t}\,\boldsymbol{\epsilon}, t\right)\right\|^2\right] \tag{18}$$

In our method, we further condition the denoising network on a scene label y, controlling the density of specific scene primitives (e.g. vegetation). The resulting conditional network $\epsilon_\theta(\mathbf{z}_{\mathcal{L}}^t, t, y)$ is trained with the modified objective:

$$\mathcal{L}_{\text{LDM}} = \mathbb{E}_{\mathbf{z}_{\mathcal{L}}^0, \boldsymbol{\epsilon}, t, y}\left[\left\|\boldsymbol{\epsilon} - \epsilon_\theta\left(\sqrt{\bar{\alpha}_t}\,\mathbf{z}_{\mathcal{L}}^0 + \sqrt{1-\bar{\alpha}_t}\,\boldsymbol{\epsilon}, t, y\right)\right\|^2\right] \tag{19}$$

B.3.3. Applications

Leveraging a latent manipulation strategy inspired by RePaint [18], our diffusion model enables practical downstream tasks without the need for additional fine-tuning.

Scene Inpainting. RePaint introduced a modified diffusion-based sampling strategy for image inpainting that employs a binary mask to distinguish known and unknown regions. The key idea is to leverage a pre-trained denoising diffusion model as a generative prior and to synchronize the denoising of unknown pixels with the noised version of known pixels. We adapt this approach to operate in the 2D latent space of a pre-trained LDM, enabling localized edits of 3D scene layouts. For simplicity, we omit the subscript $\mathcal{L}$ and denote the noisy layout latent at timestep t as $\mathbf{z}_t$. Given an initial latent $\mathbf{z}_0$, the modified denoising update is formulated as:

$$\mathbf{z}_{t-1}^{\text{known}} \sim \mathcal{N}(\sqrt{\bar{\alpha}_t}\mathbf{z}_0, (1-\bar{\alpha}_t)\mathbf{I}) \tag{20}$$

$$\mathbf{z}_{t-1}^{\text{unknown}} \sim \mathcal{N}(\mu_\theta(\mathbf{z}_t, t, y), \sigma_t^2\mathbf{I}) \tag{21}$$

$$\mathbf{z}_{t-1} = (1-\mathbf{m}) \odot \mathbf{z}_{t-1}^{\text{known}} + \mathbf{m} \odot \mathbf{z}_{t-1}^{\text{unknown}} \tag{22}$$

where $\mathbf{m}$ is a binary latent mask identifying the regions to be edited, $\odot$ denotes element-wise multiplication, and $\mathcal{N}(\mu_\theta(\mathbf{z}_t, t, y), \sigma_t^2\mathbf{I})$ represents the learned reverse process of our pre-trained diffusion model. By properly setting the mask m, we can edit an original scene in diverse ways.

Scene Outpainting. Scene outpainting extends a 3D layout beyond its original spatial extent by predicting plausible continuations of existing structures. We perform this

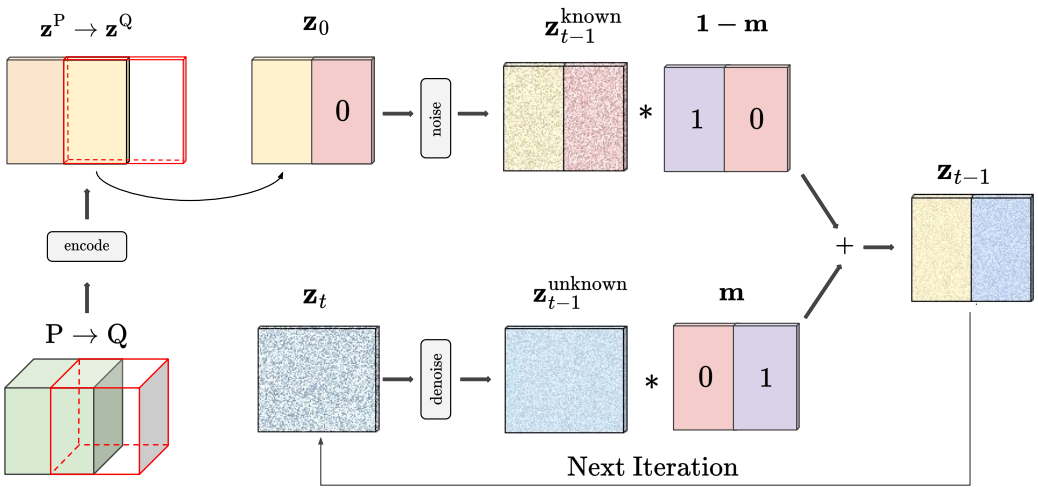


Figure 2. **Latent-space Scene Outpainting.** Given a known layout block P (green) with latent $\mathbf{z}^P$, the diffusion model predicts the latent $\mathbf{z}^Q$ for an adjacent block Q (outlined in red) that partially overlaps with P. During each sampling step, the overlapping area (yellow) is re-noised to match the current diffusion timestep, while denoising is applied to $\mathbf{z}_t$. The resulting latents are then combined through the mask $\mathbf{m}$, yielding a coherent representation $\mathbf{z}_{t-1}$ for the next iteration that preserves the known content and refines the unknown region (blue).

extrapolation directly in the latent space of our diffusion model, using the same manipulation strategy employed for inpainting. As illustrated in Fig. 2, the procedure adopts a sliding-window scheme: given a known layout block P (green) with latent representation $\mathbf{z}^P$, we synthesize the latent $\mathbf{z}^Q$ of an adjacent block Q (outlined in red) that overlaps with P. The goal is to generate $\mathbf{z}^Q$ such that the overlapping part is faithfully preserved while the extended area remains semantically and structurally aligned with P. In all experiments, we use an overlap of 50%. This choice balances structural coherence with generative freedom. A large value strengthens conditioning on the known region, promoting continuity of geometry and semantics across block boundaries, but it also restricts the model's ability to produce novel content. Conversely, overly reducing the overlap weakens the conditioning signal, often resulting in boundary artifacts and semantic misalignment. To initialize the diffusion process, we construct an input latent $\mathbf{z}_0$ by copying the overlapping region (yellow) from $\mathbf{z}^P$ and zeroing out the rest. A binary mask $\mathbf{m}$ marks the unknown area to be generated with ones and the known region with zeros. The process begins by sampling $\mathbf{z}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$, following the standard reverse diffusion formulation. During each step t, the latent $\mathbf{z}_t$ is jointly denoised across both regions, yielding an intermediate prediction $\mathbf{z}_{t-1}^{\text{unknown}}$ (see Eq. (21)). To maintain statistical compatibility between the two parts, the forward noising process is applied to the input latent $\mathbf{z}_0$, yielding $\mathbf{z}_{t-1}^{\text{known}}$ (see Eq. (20)). The two components are then merged through the mask $\mathbf{m}$ to form a unified latent representation $\mathbf{z}_{t-1}$ for the next denoising iteration (see Eq. (22)).

To expand a scene spatially, we apply this mechanism in a two-stage approach. In the first pass, we generate the four cardinal neighbors (top, bottom, left, and right), each conditioned on its corresponding overlapping region with the original layout. In the second pass, we synthesize the remaining four corner regions (top-left, top-right, bottom-left, and bottom-right), which depend on the previously generated neighbors. We note that within each pass, all blocks are generated in parallel. This two-stage process effectively doubles the spatial extent of the original layout. Iteratively applying this expansion enables the synthesis of large-scale 3D scenes, a task we refer to as large-scale scene extrapolation. For simplicity, and consistent with prior work [15], we restrict iterative extrapolation to a single spatial axis.

B.3.4. Implementation Details

All DiT variants are trained using the Diffusers library[2]. Training is performed for 3 days on 4 A100 GPUs with a batch size of 256. Optimization uses AdamW with a constant learning rate of $1e^{-4}$ and a weight decay of $1e^{-6}$. An Exponential Moving Average (EMA) with a decay rate of 0.9999 is applied to stabilize training. We employ the DDPM scheduler with 1000 timesteps and a linear variance schedule ranging from $\beta_{\text{start}} = 0.0015$ to $\beta_{\text{end}} = 0.015$. Classifier-free guidance [8] is applied with a guidance scale of 4.0. During inference, we use 250 denoising steps, while inpainting and outpainting employ a roll-back of $J = 10$ and $R = 10$ resampling iterations.

C. Experimental Details

In this section, we describe the synthetic dataset and evaluation protocol used to compare our Cholesky-based encoding

[2] https://github.com/huggingface/diffusers

Method	Hierarchical	Level	Voxel Size (m)	Voxel Resolution	VAE Train (days)	Diff. Train (days)
SemCity [13]	✗	Single	0.25	$256^2 \times 32$	3	3
PDD [15]	✓	Coarse	2.0	$32^2 \times 4$	—	1
		Medium	1.0	$64^2 \times 8$	—	2
		Fine	0.25	$256^2 \times 16$	—	14
XCube [20]	✓	Coarse	1.0	$64^2 \times 8$	1	2
		Fine	0.25	$256^2 \times 32$	6	7
PrITTI (Ours)	✗	Single	—	—	3	3

Table 4. **Comparison of Architectures and Training Configurations.** For each method, we report the model hierarchy, resolution settings, and the corresponding VAE and diffusion training times (where applicable).

with a quaternion-based alternative. For this ablation study, we generate synthetic datasets of increasing size, where each sample contains a single zero-centered vehicle primitive with randomly sampled scales and yaw rotations. The training sets consist of 1.2K, 2.5K, 5K, and 10K samples, with each larger set including all samples from the smaller ones. A shared test set of 2K samples is used for evaluation. To reduce variance from random sampling, dataset generation is repeated with five different random seeds. For each seed and training size, both encoding variants are trained for 20 epochs, and the mean IoU3D is computed on the shared test set. Final results are reported as the average IoU3D over all seeds for each dataset size.

D. Baseline Methods

To the best of our knowledge, no prior work tackles 3D semantic urban layout generation using primitives. We therefore compare against three recent and well-established voxel-based methods, all trained from scratch on our voxelized dataset: SemCity [13], PDD [15], and XCube [20]. This comparison allows us to empirically demonstrate the benefits of primitives over voxels, the prevailing representation in this field, which is central to our contribution. SemCity adopts a continuous triplane diffusion model trained on the SemanticKITTI [2] and CarlaSC [24] datasets. It first trains a triplane autoencoder and then applies diffusion within the learned latent triplane space. The model follows a non-hierarchical design and supports downstream tasks such as semantic scene inpainting, outpainting, and completion refinement. PDD employs a discrete, hierarchical diffusion paradigm trained on the synthetic CarlaSC dataset. It generates scenes in a coarse-to-fine manner across three pyramid levels. The first level synthesizes scenes unconditionally from noise, while subsequent levels condition on the preceding outputs to progressively refine scene details. PDD supports large-scale scene outpainting and conditioned scene generation, the latter refining a ground-truth coarse layout into a more detailed semantic scene. XCube generates large-scale 3D scenes using a hierarchical sparse-voxel representation trained across the Karton City [1] and Waymo [22] datasets. Each level in the hierarchy is trained independently and consists of a VAE-diffusion pair trained sequentially (VAE first, then diffusion). Both the VAE and diffusion components use 3D UNet backbones. For urban scenes, XCube supports both unconditional and single-LiDAR-scan–conditioned generation, though the conditional implementation is not yet publicly available. Tab. 4 summarizes the model and training configurations for all baseline methods and PrITTI. Following the original implementations, PDD and XCube are trained using 4 and 8 A100 GPUs, respectively. In addition to training configurations, we also compare the peak GPU memory usage of all methods under the same inference setup. As shown in Tab. 5, our method is more memory-efficient than voxel-based baselines, consistent with the use of a primitive-based representation.

Metric	SemCity	PDD	XCube	Ours (DiT-B)
Peak GPU Memory (GB)	8.30	1.33	4.84	**0.89**

Table 5. Peak GPU memory usage (GB) under identical inference settings (100 scenes, batch size 1, single GPU).

E. Additional Qualitative Results

In this section, we present additional qualitative results for all tasks described in the main paper.

3D Semantic Scene Generation. Fig. 8 presents 3D semantic scenes generated by our method. By conditioning the LDM on discrete scene labels controlling vegetation density, we can produce scenes with (a) high, (b) medium, and (c) low vegetation. Additional BEV semantic renderings of synthesized scenes, used for computing the generative metrics reported in the main paper, are shown in Fig. 9. Notably, conditioning on vegetation density serves only as a minimal illustration of controllability through our class-conditional DiT. This conditioning scheme is general and can be naturally extended to other primitive categories (e.g., vehicles) or their combinations. To demonstrate this, we additionally trained models conditioned on vehicle density and joint

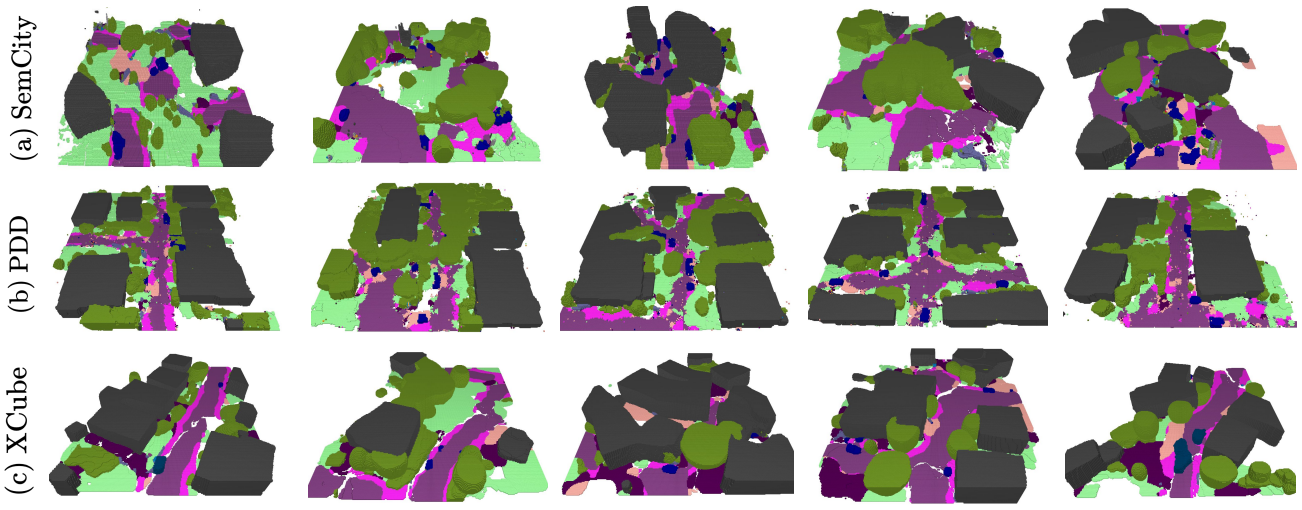


Figure 3. **Low-quality scene generation examples** from baseline methods exhibiting fragmented geometry and semantic inconsistencies.

vegetation–vehicle densities. Qualitative results are shown in Fig. 10 and Fig. 11, respectively. For comparison, we provide 3D and BEV rendering results for all baselines, generated unconditionally: SemCity (Fig. 12, Fig. 13), PDD (Fig. 14, Fig. 15), and XCube (Fig. 16, Fig. 17). As seen in their BEV renderings and Fig. 3, these methods often produce scenes with limited semantic consistency and structural coherence, exhibiting blended geometries, irregular object shapes, and noisy semantic predictions. These artifacts reduce the perceived realism of the results, aligning with the lower quantitative generative performance in the main paper.

Scene Inpainting and Outpainting. Fig. 18 and Fig. 21 present additional scene inpainting and outpainting results produced by our method, respectively. As illustrated in Fig. 18, by appropriately setting the binary latent mask $\mathbf{m}$ (see Eq. (22)), we can achieve flexible, localized scene edits in arbitrary directions (e.g. top, bottom, left, right). Among the baselines, only SemCity reports both inpainting and outpainting results. Fig. 20 compares our method with SemCity, demonstrating that PrITTI produces coherent and semantically consistent edits and extensions, whereas SemCity often yields fragmented or contextually inconsistent structures, particularly in the outpainting setting. XCube does not address these scene-level editing tasks, and PDD provides only qualitative results for large-scale scene extrapolation, which we compare against in a later section.

Ground-conditioned 3D Primitives Generation. Fig. 19 illustrates a special case of inpainting enabled by our disentangled latent representation. By keeping the ground channels fixed and inpainting only the object channels, our model generates novel object primitives conditioned on a consistent ground layout. This is achieved using a pre-trained LDM without any additional training.

Large-scale Scene Extrapolation. Fig. 22 shows large-scale scene extrapolation results of our method, starting from an initially generated layout block (outlined in red) and progressively extending it upward. The extrapolated layouts exhibit strong global coherence, preserving smooth road connectivity and semantically consistent transitions between synthesized regions. All examples use the medium vegetation density condition. Changing the label to low or high can enable extrapolation of the same initial block under different vegetation settings, highlighting our model's controllability (see supplementary video for an example). For comparison, we also include results from PDD, the only baseline supporting large-scale scene extrapolation. SemCity provides code only for outpainting a generated block twice (see Fig. 20), and therefore cannot be evaluated here. As shown in Fig. 23, PDD often fails to maintain global structure: roads are disconnected between adjacent blocks, buildings are frequently misplaced, and abrupt semantic transitions occur across generated areas, resulting in overall fragmented and visually implausible large-scale layouts. In contrast, our method produces continuous, well-aligned extrapolations with consistent semantics and geometry across large regions.

Object-level Editing. Our framework maintains an instance-level vector representation for each primitive, parameterized by its 3D center location and 6D Cholesky parameters. This formulation allows intuitive manipulation of individual objects by directly updating their corresponding parameters (e.g. center for translation, Cholesky parameters for rotation and scaling). Fig. 24 illustrates additional editing results, where simple transformations are applied to randomly selected vehicle primitives from the original scene (column a). Columns (b-e) visualize examples of dropout, rotation, scaling, and translation, respectively. In contrast, object-

level manipulation in voxel-based scene representations remains highly challenging. First, identifying which voxels correspond to a particular object typically requires explicit segmentation, which is often complicated by ambiguous object-background boundaries. Furthermore, even after segmentation, applying a transformation such as translation requires moving the corresponding voxel subset and then filling in the vacated space. Consequently, existing voxel-based approaches for 3D semantic urban scene generation provide little or no support for object-level editing. For instance, PDD and SemCity do not enable any form of direct instance-level manipulation, while XCube only supports manual voxel addition or removal for a single object at the coarsest level of its hierarchy through a dedicated user interface.

Photo-realistic Street View Synthesis. Fig. 25 presents additional examples demonstrating how 2D semantic maps rendered from our generated 3D scenes can be translated into photo-realistic street-view images. For this task, we employ the ControlNet [25] model from Urban Architect [16], which is fine-tuned on KITTI-360 3D semantic layouts. Although some visual artifacts exist, the rendered maps serve as effective conditioning signals and reliably guide the image generation process. Notably, despite the coarse geometry of our primitives, the synthesized images reflect object shapes beyond cuboids or ellipsoids. We emphasize that photo-realistic appearance synthesis is not the primary objective of our work but rather an additional downstream application.

F. Precision and Recall Metrics

To evaluate our generative models, among others, we use the improved precision and recall metrics introduced by Kynkäänniemi et al. [12]. Let $\mathbf{\Phi}_r$ and $\mathbf{\Phi}_g$ denote the feature sets extracted from real and generated samples, respectively. A binary function $f(\boldsymbol{\phi}, \mathbf{\Phi})$ determines whether a feature vector $\boldsymbol{\phi}$ lies within the estimated manifold of a reference set $\mathbf{\Phi} \in \{\mathbf{\Phi}_r, \mathbf{\Phi}_g\}$:

$$f(\boldsymbol{\phi}, \mathbf{\Phi}) = \begin{cases} 1, & \text{if } \|\boldsymbol{\phi} - \boldsymbol{\phi}'\|_2 \leq \|\boldsymbol{\phi}' - \mathrm{NN}_k(\boldsymbol{\phi}', \mathbf{\Phi})\|_2, \\ & \text{for at least one } \boldsymbol{\phi}' \in \mathbf{\Phi}, \\ 0, & \text{otherwise.} \end{cases} \tag{23}$$

Here, $\mathrm{NN}_k(\boldsymbol{\phi}', \mathbf{\Phi})$ denotes the k^{th} nearest neighbor of $\boldsymbol{\phi}'$ within $\mathbf{\Phi}$. Using this function, precision and recall are computed as:

$$\text{precision}(\mathbf{\Phi}_r, \mathbf{\Phi}_g) = \frac{1}{|\mathbf{\Phi}_g|} \sum_{\boldsymbol{\phi}_g \in \mathbf{\Phi}_g} f(\boldsymbol{\phi}_g, \mathbf{\Phi}_r) \tag{24}$$

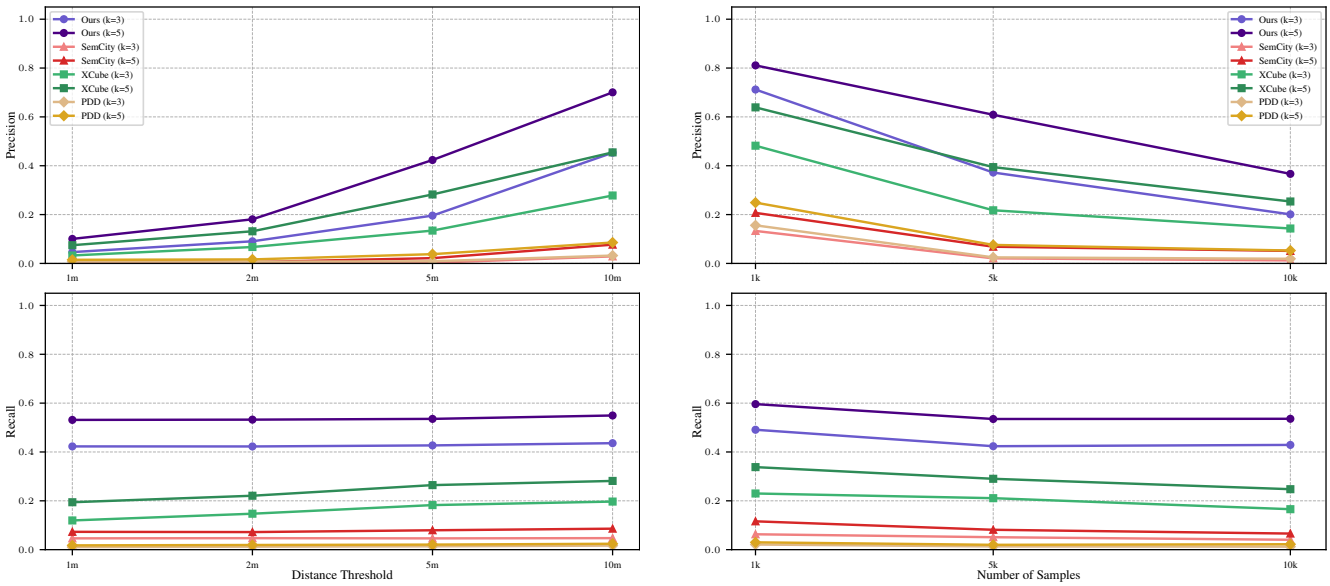
$$\text{recall}(\mathbf{\Phi}_r, \mathbf{\Phi}_g) = \frac{1}{|\mathbf{\Phi}_r|} \sum_{\boldsymbol{\phi}_r \in \mathbf{\Phi}_r} f(\boldsymbol{\phi}_r, \mathbf{\Phi}_g) \tag{25}$$

Intuitively, precision measures the proportion of generated samples that appear realistic (i.e. fall within the manifold of real data), whereas recall quantifies how well the generated samples cover the diversity of real samples. Following [12], we use an equal number of real and generated samples ($|\mathbf{\Phi}_r| = |\mathbf{\Phi}_g|$). However, we found out that these metrics are highly sensitive to several factors including: **(i)** implementation details, **(ii)** the diversity of real samples, **(iii)** the number of evaluation samples, and **(iv)** the neighborhood size k used for manifold estimation (see Eq. (23)). For **(i)**, we follow [19], which forms the basis of our DiT models, and use the evaluation suite from [7] for consistency. To address **(ii)**, we implement two separate strategies for sampling real poses. The first enforces a minimum spatial distance (in meters) between neighboring poses. The second, inspired by farthest point sampling (FPS), selects poses sequentially to maximize their distance from previously chosen ones. The number of evaluation samples **(iii)** depends on the applied sampling strategy. In the distance-based approach, the sample size is implicitly determined by the distance constraint, whereas in FPS we explicitly control it, evaluating with 1K, 5K, and 10K real samples. Finally, to study **(iv)**, we vary the neighborhood size $k \in \{3, 5\}$, examining its effect on the generative performance. Comprehensive evaluations of precision and recall under these configurations are presented in Fig. 4, comparing our approach against SemCity, XCube, and PDD. For PDD and XCube, results are reported at their finest resolution, and for SemCity we use the 1M variant.

For both sampling strategies, we observe a consistent decrease in precision as the number of evaluation samples increases. In the distance-based strategy, this effect corresponds to smaller distance thresholds (e.g. $1\,m$ yields 32,203 samples, whereas $10\,m$ yields 3,855). This behavior follows Eq. (23): as more real samples are included, the hyperspheres defined by their k^{th} nearest neighbors become smaller and more densely packed. Consequently, even visually realistic generated samples are less likely to fall within any real sample's hypersphere. In contrast, recall remains relatively stable across varying sample counts, likely because denser sampling refines the real manifold without altering its overall extent. Finally, both precision and recall increase with larger neighborhood sizes k. This trend follows directly from Eq. (23): increasing k expands each sample's hypersphere, thereby loosening the inclusion criterion.

G. Overfitting Analysis

To assess potential memorization of dataset-specific patterns, we perform a nearest-neighbor (NN) analysis in feature space. We extract features using a pretrained VGG16 network and compute Euclidean NN distances between 50K generated BEV samples and a fixed subset of 50K training samples. To contextualize these values, we additionally compute real-to-real distances on the same subset by match-



(a) Distance Threshold Sampling

(b) Farthest Pose Sampling

Figure 4. **Precision and recall comparison** between our method (using DiT-B), SemCity, XCube, and PDD under two evaluation protocols: (a) distance-threshold reference-pose sampling and (b) farthest-pose sampling, each tested across varying neighborhood sizes.

ing each training sample to its nearest neighbor excluding self-matches. We observe that generated samples are consistently farther from the training set (mean NN distance ≈ 52.2, median ≈ 51.7, minimum ≈ 27.2, max ≈ 115.7) than real samples are from each other (mean NN distance ≈ 22.5, median ≈ 22.0, minimum ≈ 4.9, max ≈ 58.7). This suggests that the model does not simply memorize or reproduce training examples. This is further supported by the qualitative examples in Fig. 5, where generated samples differ from their three nearest neighbors in the training set in layout and object arrangement.

H. Applicability beyond KITTI-360

We conduct our main experiments on KITTI-360, which, to the best of our knowledge, is the only real-world dataset providing primitive-level annotations for entire urban scenes (e.g. buildings, vegetation) rather than focusing on selected object classes (e.g. vehicles). This yields scenes with many primitives and thus a demanding setting for training and evaluation. Importantly, PrITTI does not require *dense* primitive annotations, but only parametric 3D cuboids or ellipsoids for arbitrary semantic categories. When fewer categories are annotated, the model simply predicts fewer primitives overall, which makes the task easier. Many autonomous driving datasets [3, 4, 23] already provide such cuboid annotations. Training PrITTI on any of them involves grouping their labels into dataset-specific primitive categories and converting the cuboid parameters into our vectorized representation. For ground geometry, 3D HD maps (e.g. in [23]) enable BEV

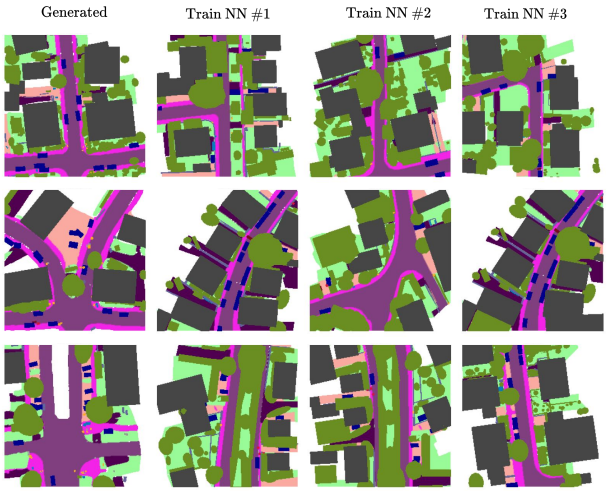


Figure 5. **Nearest-neighbor Analysis.** For each generated sample (left), we show its closest training samples (right). Generated scenes differ in layout and object placement, suggesting the model does not memorize training data.

height fields rendering, while 2D maps (e.g. in [3]) can be handled via a flat-ground assumption.

To demonstrate applicability beyond KITTI-360, we additionally train our method on Argoverse 2 (AV2) [23]. We group the original AV2 labels into seven primitive categories: vehicle big, vehicle small, two-wheelers, human, pole, traffic control, and object. We emphasize that PrITTI does not require a fixed category set across datasets, and for AV2 these groups happen to align well with a subset of those used

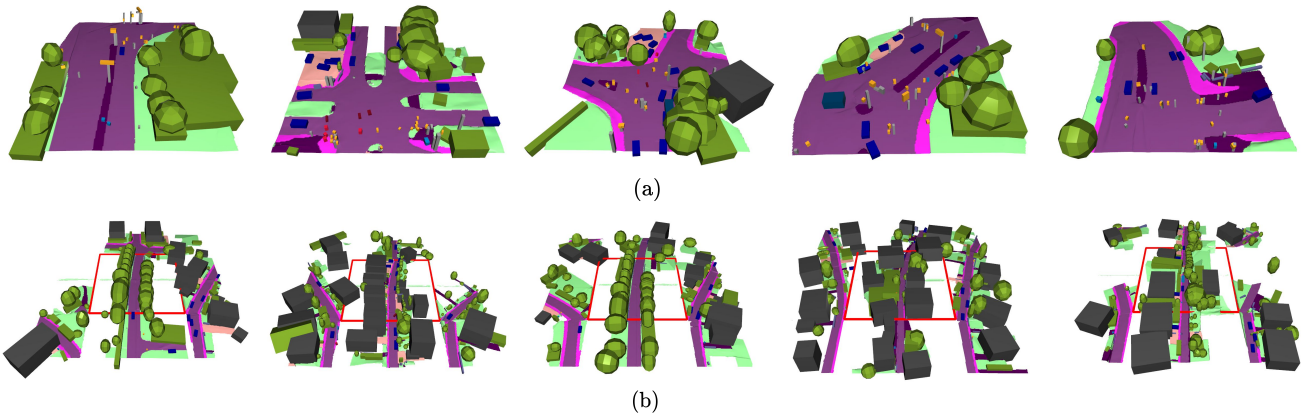


Figure 6. **PrITTI Failure Cases.** (a) Fine-grained attachments (e.g. pole-mounted lamps) are often not accurately reconstructed, yielding floating objects. (b) Large empty conditioning regions in outpainting can lead to weak or inconsistent completions.

Raster ($\times 10^{-2}$)		Primitive	
MSE ↓	**IoU** ↑	**AP3D** ↑	**AP3D@50** ↑
0.015	99.98	56.495	39.490

(a) Reconstruction

Precision ↑	**Recall** ↑	**FID** ↓	**IS** ↑
0.572	0.410	144.330	2.878

(b) Generation (DiT-B)

Table 6. PrITTI evaluation results on AV2 [23], showing (a) reconstruction and (b) generation metrics.

in KITTI-360 (see Tab. 1). Notably, we find from dataset statistics that only 84 total queries are required, compared to 514 for [14], highlighting the reduced task complexity. We report qualitative generation results in Fig. 7 and quantitative results for both reconstruction and generation in Tab. 6.

I. Limitations and Failure Cases

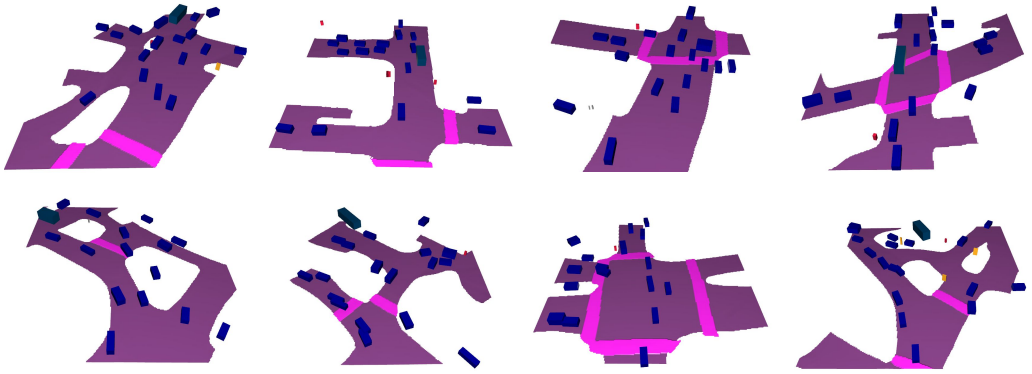
Despite its promising results, our approach comes with certain limitations. First, as discussed in the main paper, the current framework is limited to static scenes and represents objects using coarse 3D primitives, which reduces geometric fidelity. Second, the model operates on a fixed set of semantic categories. Although this aligns with prior semantic layout methods [13, 15, 20, 26], it limits the generation of unseen object types. Extending the framework toward open-vocabulary scene generation is an exciting future direction. In addition, ground elements are modeled as rasterized 2D representations that are subsequently extruded into 3D. Compared to voxel-based approaches that maintain a 3D grid over all scene elements, using a 2D representation yields better memory and computational scalability. However, the extrusion process is still inherently constrained by the resolution of the input raster maps, potentially resulting in less detailed ground surfaces. Furthermore, accurately reconstructing fine-grained attachments (e.g. lamps or traffic lights mounted on poles) and small, densely arranged structures (e.g. fences) remains quite challenging. Such geometric inaccuracies can be learned by the autoencoder and propagated to the diffusion model, occasionally producing scenes with floating or disconnected objects (see Fig. 6 (a)). Moreover, our framework does not explicitly enforce hard geometric constraints and instead relies on training data to capture such relationships. While the generated scenes are generally plausible, occasional geometric inconsistencies arise, especially in crowded scenes. In particular, primitives may intersect, which can sometimes be valid (e.g. vehicles with attached trailers) but not always appropriate (e.g. two vehicles placed unrealistically close to each other). Similarly, ground-primitive misalignment can produce semantically inconsistent configurations, such as buildings intersecting the drivable road surface (Fig. 9, right). Incorporating soft constraints or geometry-aware guidance during diffusion [17] could mitigate these issues and improve physical validity. Finally, in the outpainting setting, failures may occur when the conditioning block contains large empty regions along certain extrapolation directions. In such cases, the diffusion model is afforded greater generative freedom, which can lead to weakly connected or semantically inconsistent completions (see Fig. 6 (b)). Additionally, some visible seams can appear across block transitions in the ground surfaces.

J. Potential Societal Impacts

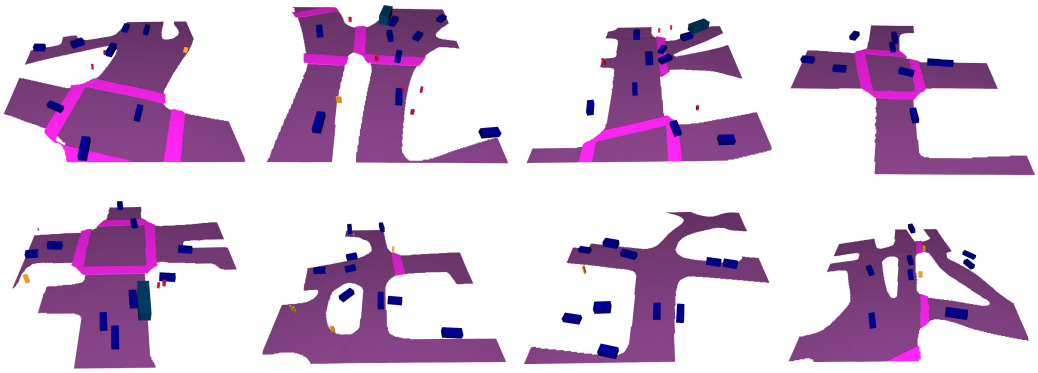
Our work addresses 3D semantic scene generation for urban environments using primitive-based representations. Traditional approaches to constructing such 3D environments require expert knowledge and significant manual effort, mak-

ing them costly and time-consuming. Moreover, reliance on pre-existing layout data, which remains scarce, particularly for outdoor scenes, limits diversity and leads to repetitive scene compositions. In contrast, our method enables the automatic generation of diverse and realistic 3D semantic scenes. This has the potential to benefit a range of applications in simulation, robotics, and autonomous driving. By providing interpretable and controllable 3D content, it can contribute to safer and more scalable environments for testing autonomous systems and improved scene understanding for embodied agents. Nevertheless, as with all generative models, there is potential for misuse. Synthetic scenes could, for instance, be employed to misrepresent real urban layouts, simulate favorable outcomes in planning proposals, or test vision-based monitoring systems (e.g. for traffic detection) under unrealistic conditions. Furthermore, our primitive-based representation simplifies real-world geometry, which, if over-relied upon, may lead to misleading conclusions in downstream tasks. These risks, however, do not diminish the framework's practical value. As with existing modeling and simulation tools, effective use relies on a clear understanding of the method's assumptions and limitations.

(a) High Vehicles



(b) Medium Vehicles



(c) Low Vehicles

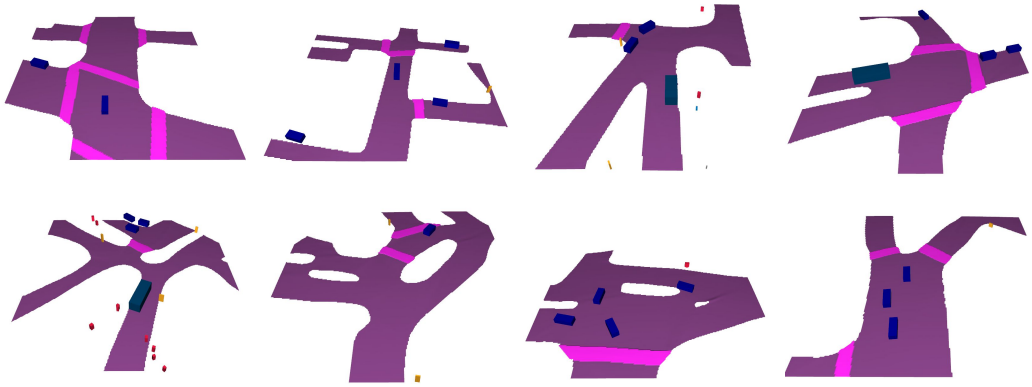
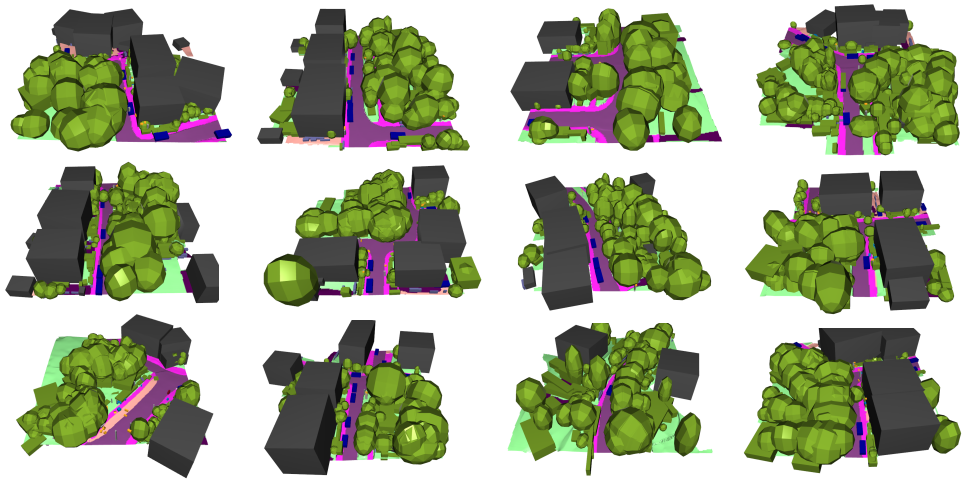
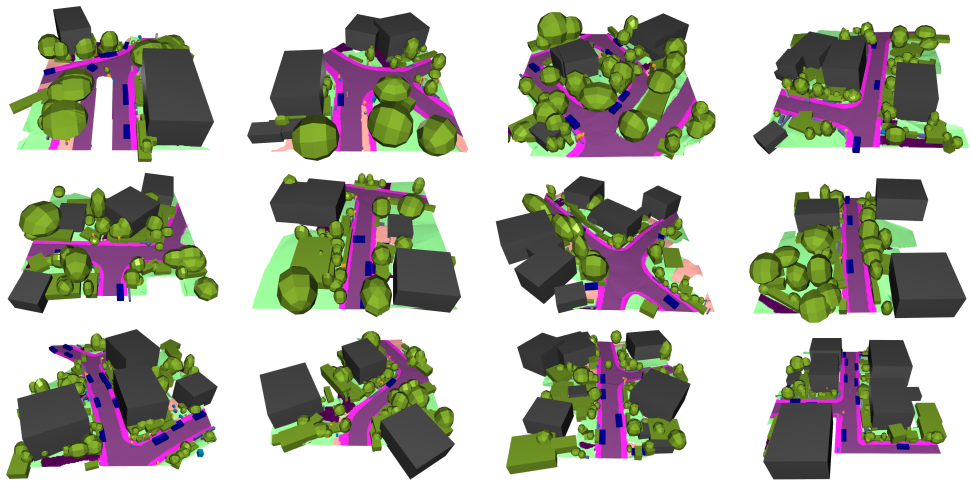


Figure 7. **PrITTI Scene Generation Results on Argoverse 2 (AV2) Dataset.** Generated 3D semantic scenes grouped under three vehicle-density conditions: (a) high, (b) medium, and (c) low. Vehicles appearing outside the drivable surface correspond to parked cars, as AV2 [23] HD maps do not provide explicit parking area annotations, unlike KITTI-360.

(a) High Vegetation



(b) Medium Vegetation



(c) Low Vegetation

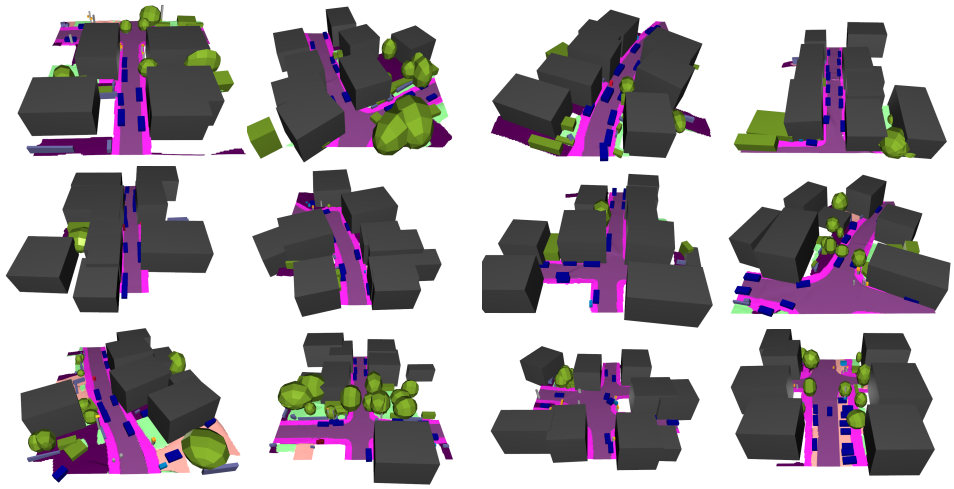


Figure 8. **PrITTI Scene Generation Results.** Generated 3D semantic scenes grouped under three vegetation-density conditions: (a) high, (b) medium, and (c) low. Across all settings, PrITTI produces diverse and realistic urban layouts with coherent ground surfaces and well-shaped primitives placed in semantically appropriate locations.

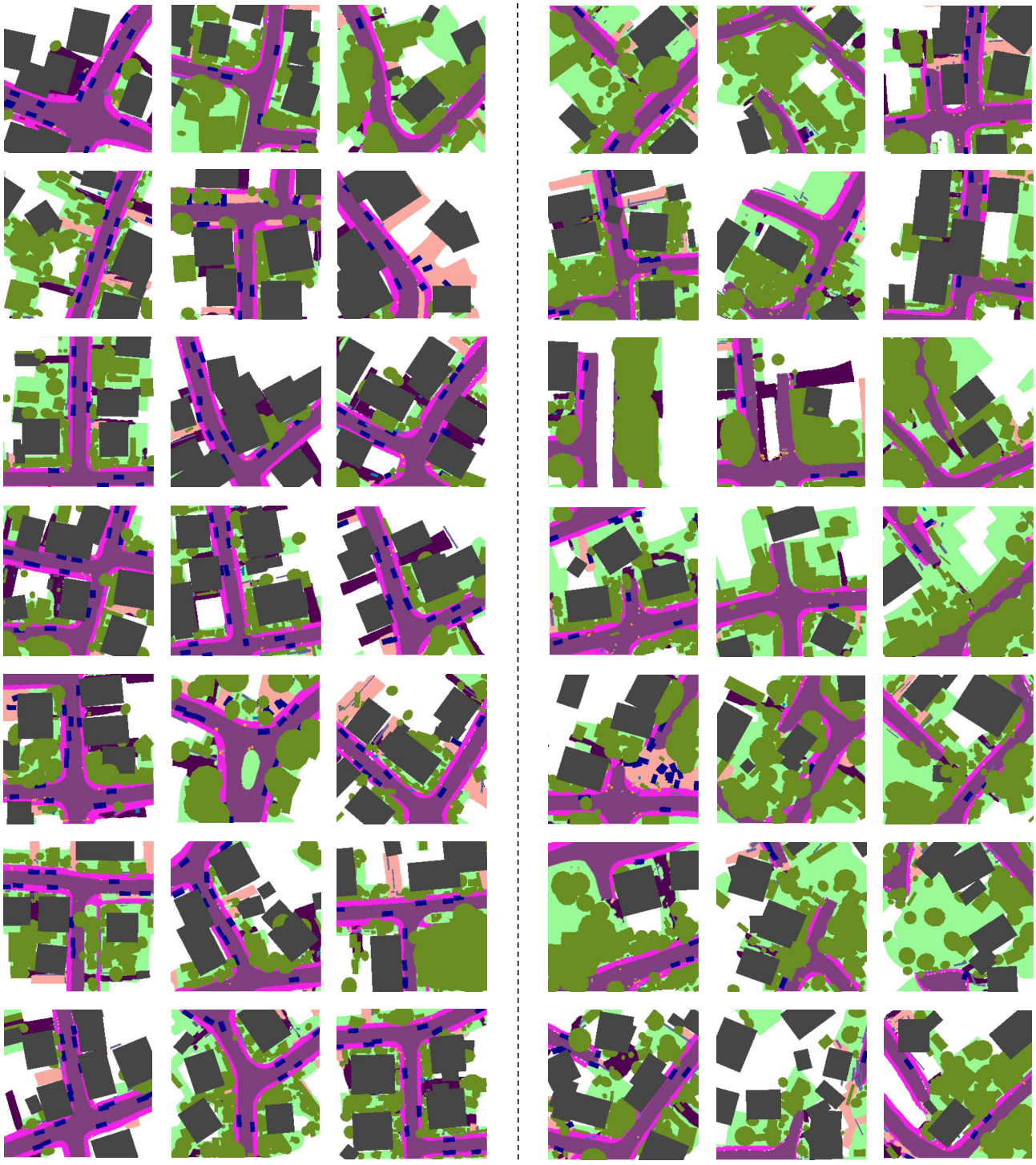


Figure 9. **BEV Renderings of PrITTI-generated Scenes.** Examples on the left illustrate well-structured and realistic scene generations, while those on the right show poorly synthesized samples with geometric or semantic inconsistencies.

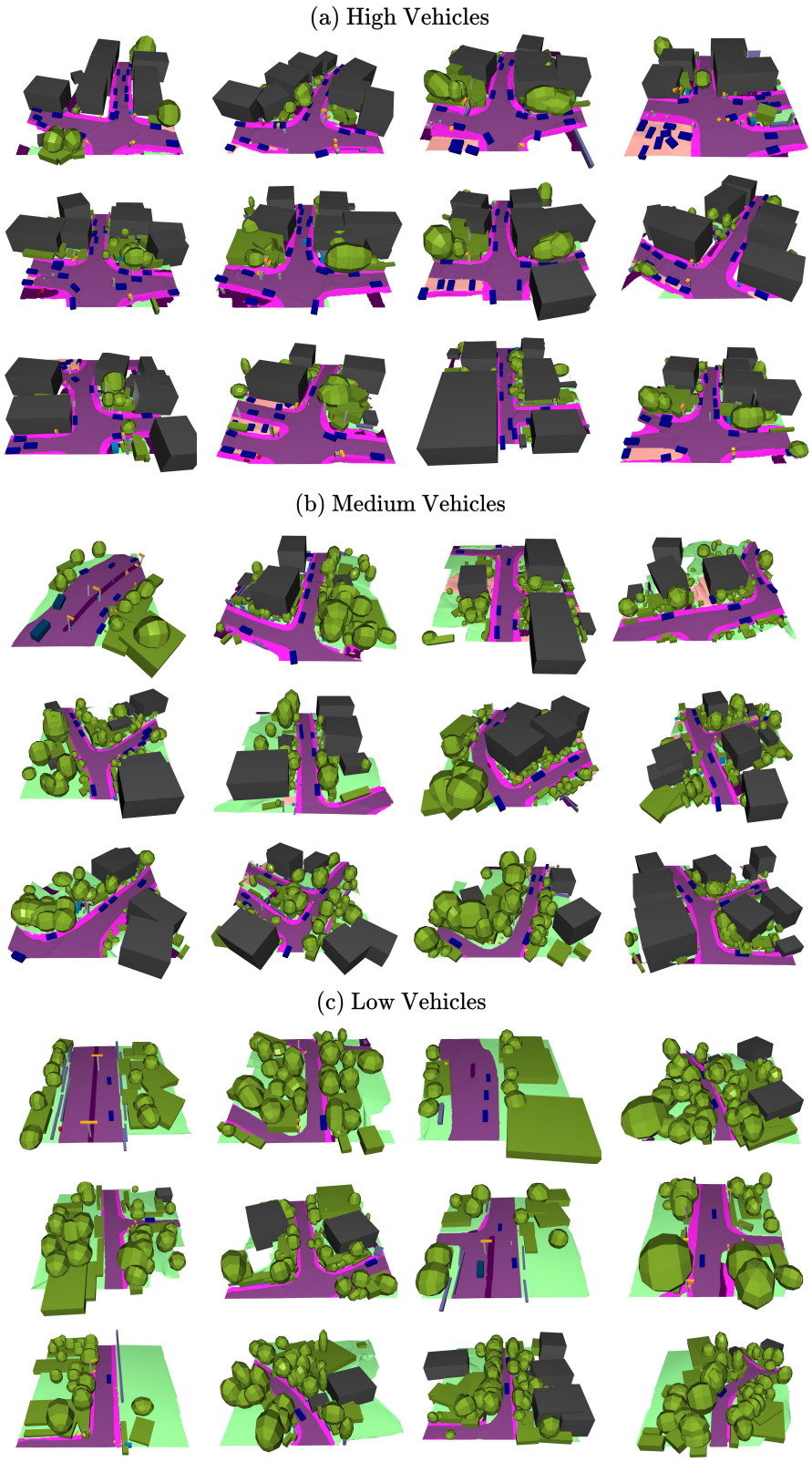
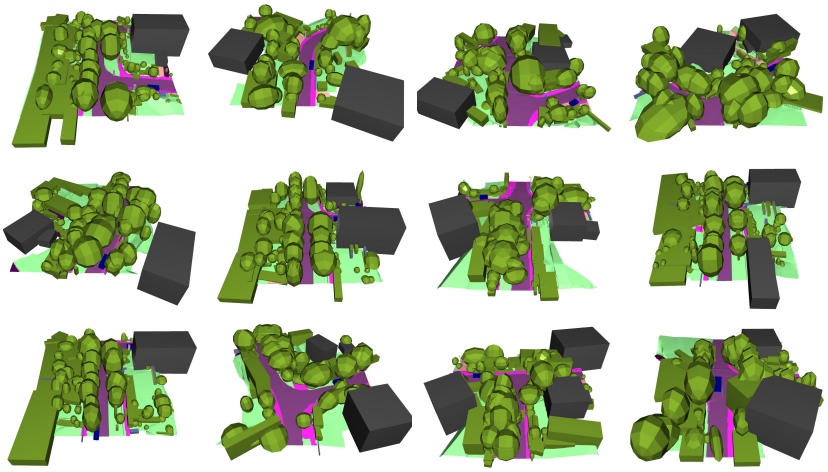
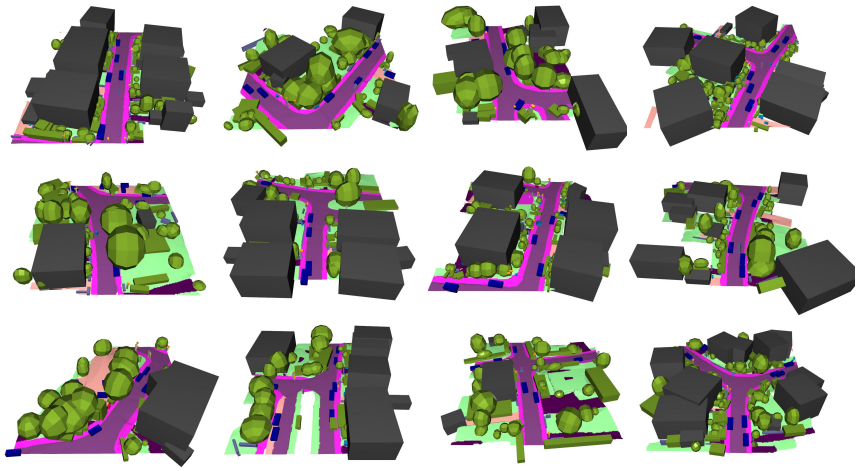


Figure 10. **PrITTI Vehicle Density-Conditioned Scene Generation.** Generated 3D semantic scenes grouped under three vehicle-density conditions: (a) high, (b) medium, and (c) low, demonstrating controllability over primitive categories beyond vegetation.

(a) High Vegetation, Low Vehicles



(b) Medium Vegetation, Medium Vehicles



(c) Low Vegetation, High Vehicles

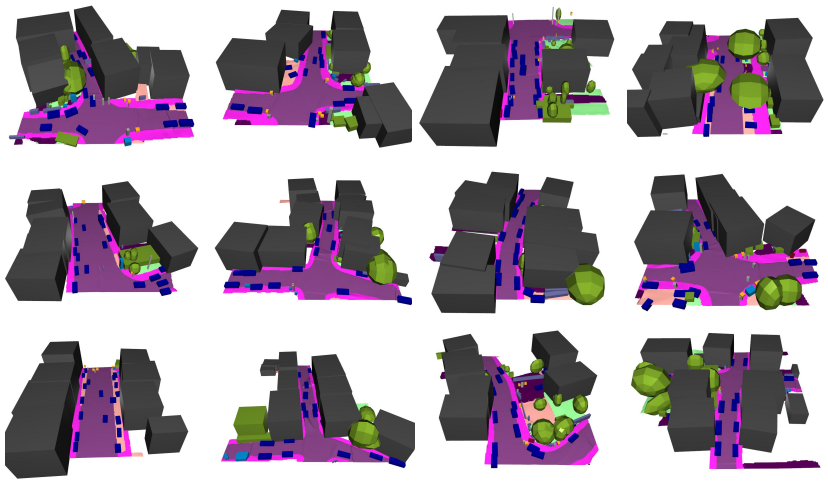


Figure 11. **PrITTI Joint Vegetation–Vehicle Density-Conditioned Scene Generation.** Generated scenes conditioned on combinations of vegetation and vehicle densities. From the nine possible settings, we show three examples: (a) high vegetation, low vehicles, (b) medium vegetation, medium vehicles, and (c) low vegetation, high vehicles, illustrating controllability over combinations of primitive categories.

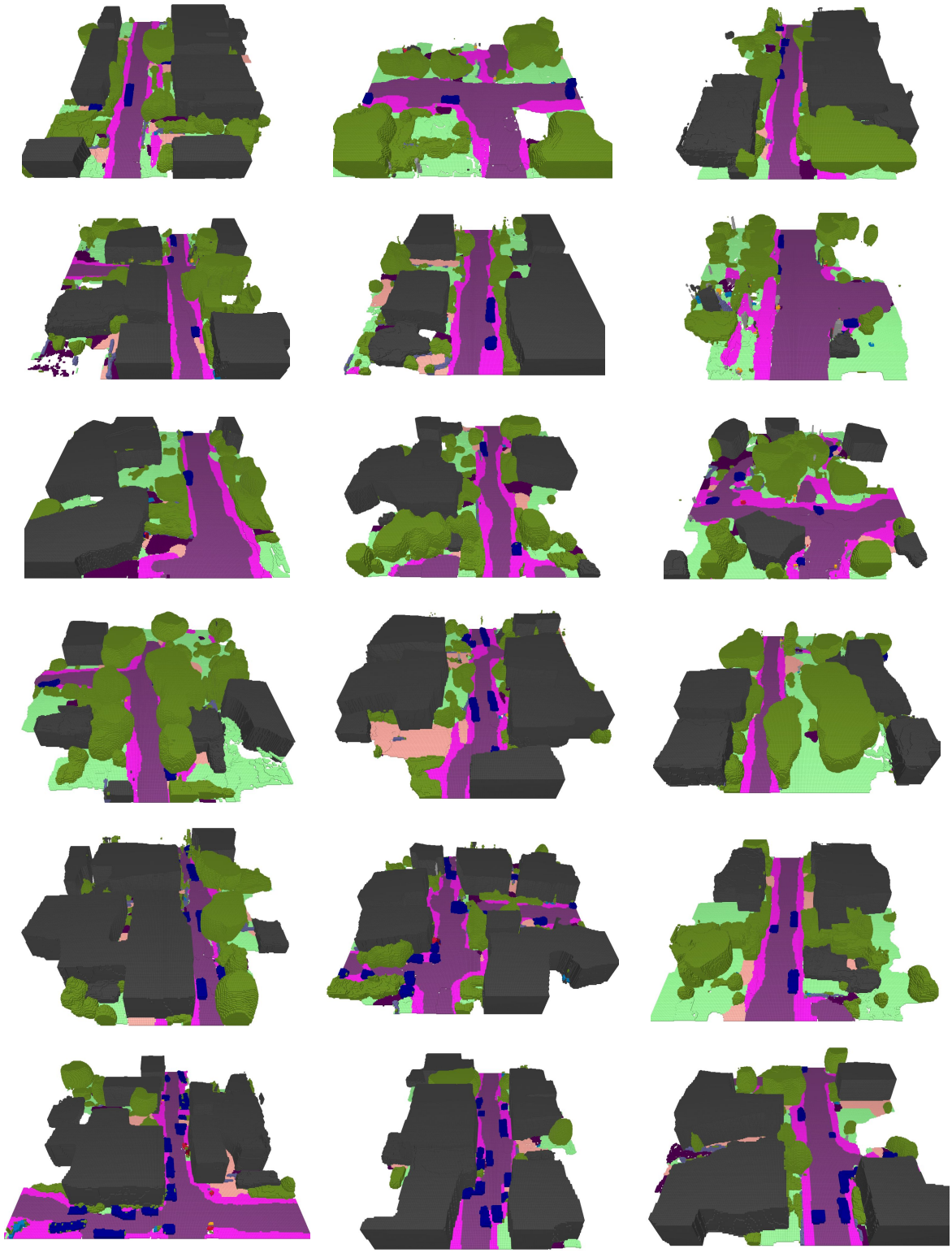


Figure 12. **SemCity Scene Generation Results.** Unconditional 3D semantic scene samples generated by SemCity [13], occasionally exhibiting irregular object shapes, blurred boundaries, and inconsistent semantic structures.

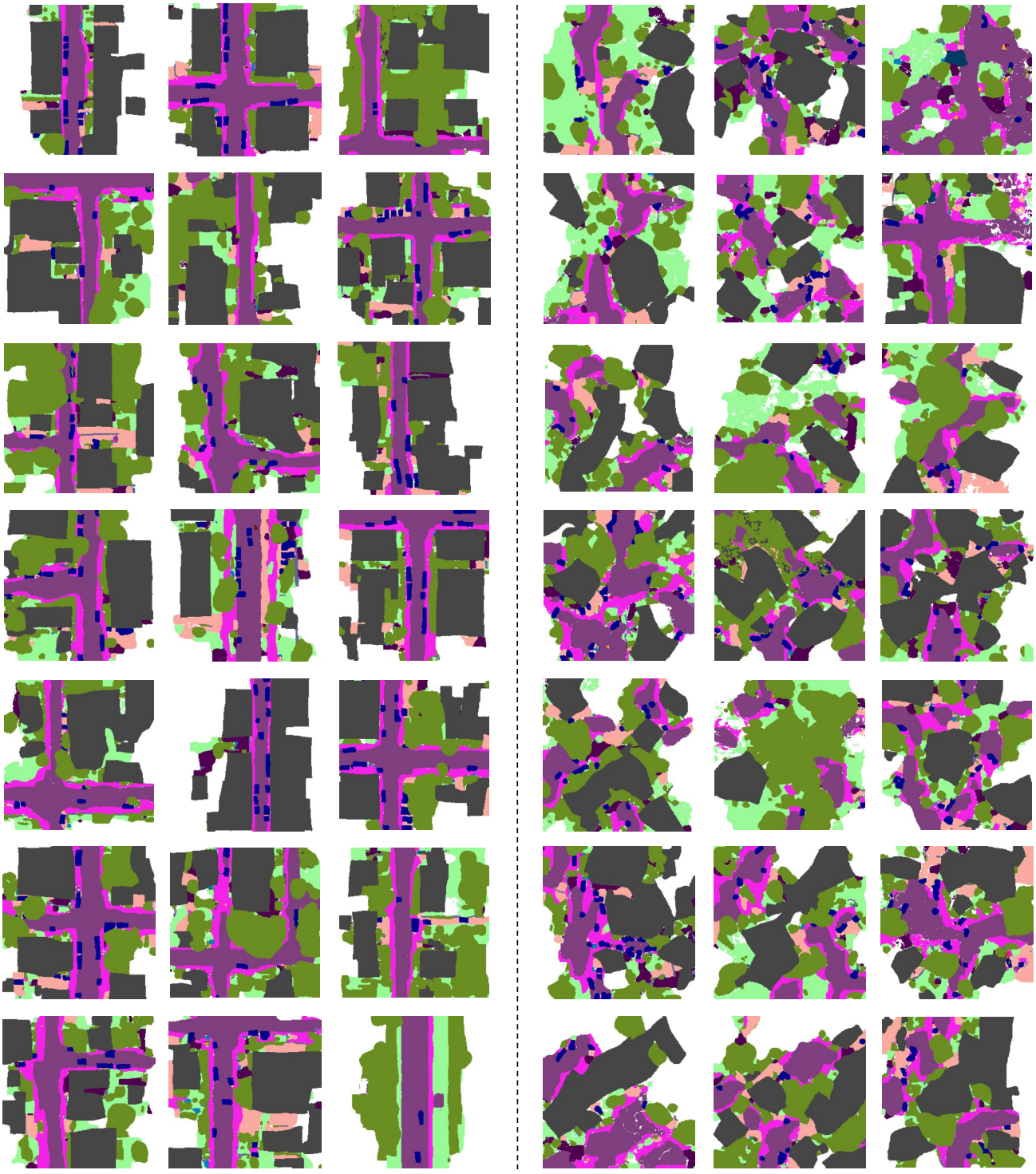


Figure 13. **BEV Renderings of SemCity-generated Scenes.** The examples on the left show relatively well-structured and semantically coherent layouts, while those on the right exhibit fragmented, irregular geometry and inconsistent semantics.

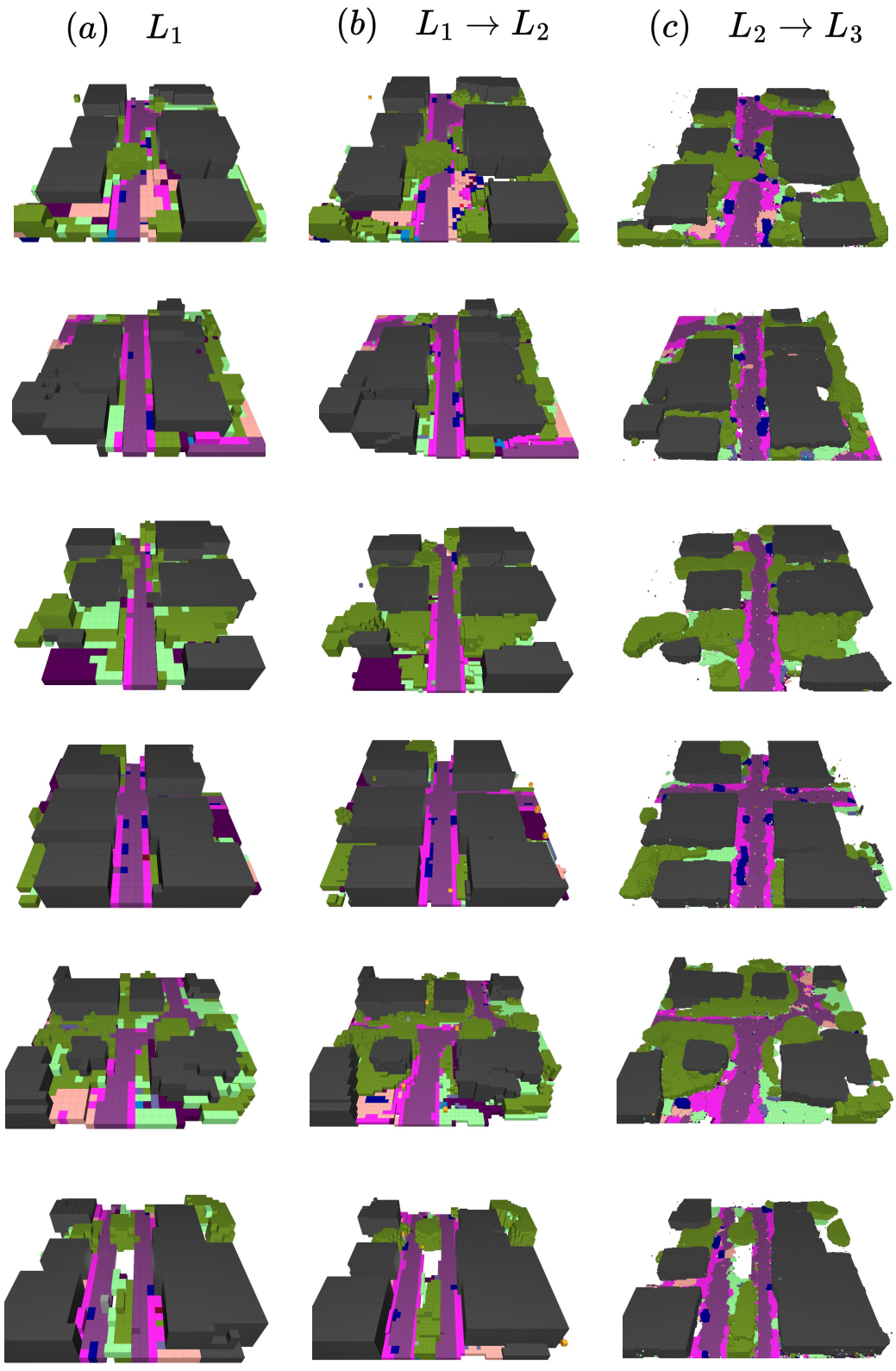


Figure 14. **PDD Hierarchical Scene Generation Results.** Unconditional 3D semantic scenes generated by PDD [15] at successive pyramid levels. While scene details increase across levels, the final stage also introduces noticeable noise and semantic inconsistencies.

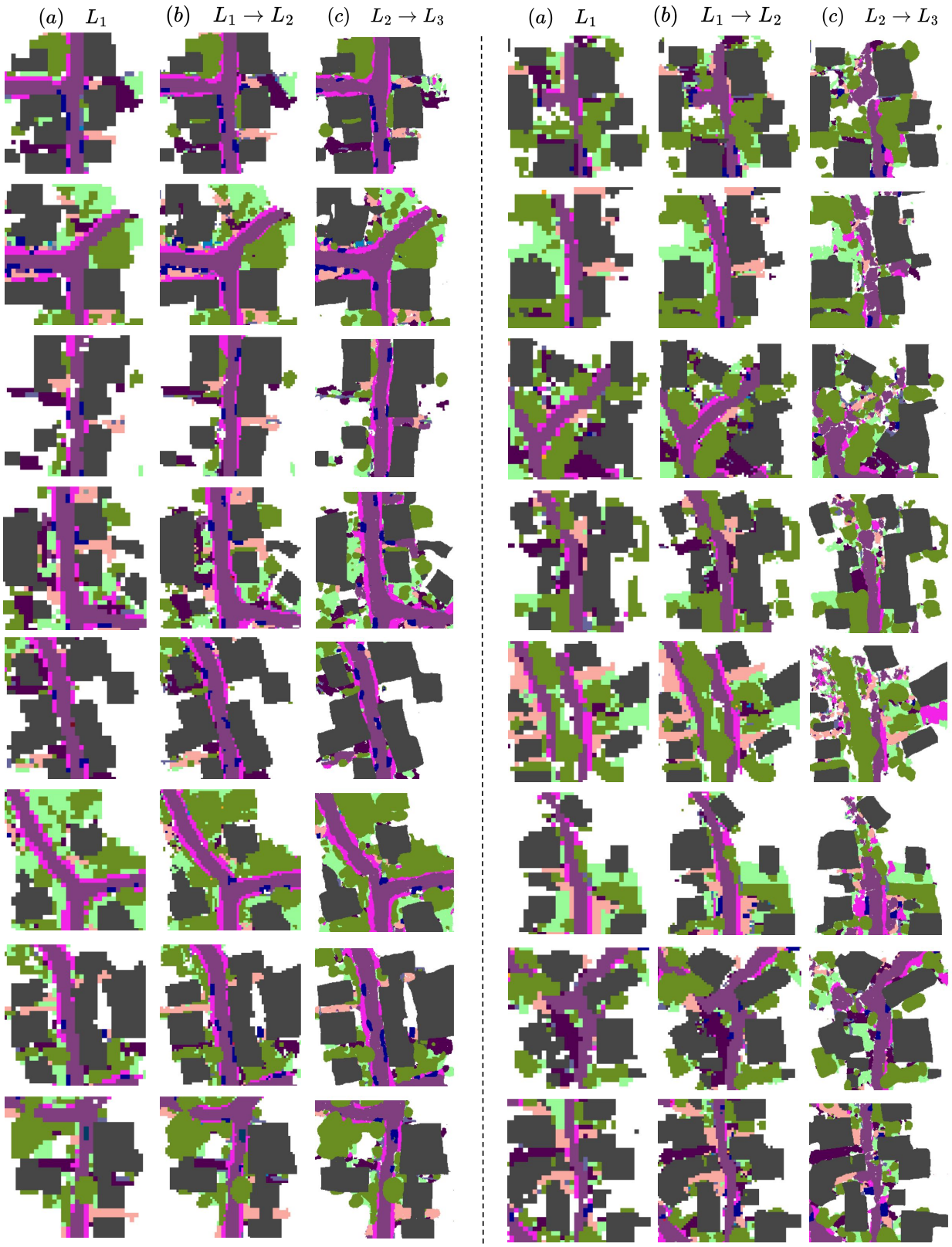


Figure 15. **Hierarchical BEV Renderings of PDD-generated Scenes.** Each column triplet shows results across successive hierarchy levels. Examples on the left exhibit relatively coherent scene structure, whereas those on the right clearly show that the last refinement stage introduces noticeable noise and semantic inconsistencies, thereby reducing the perceived realism of the final outputs.

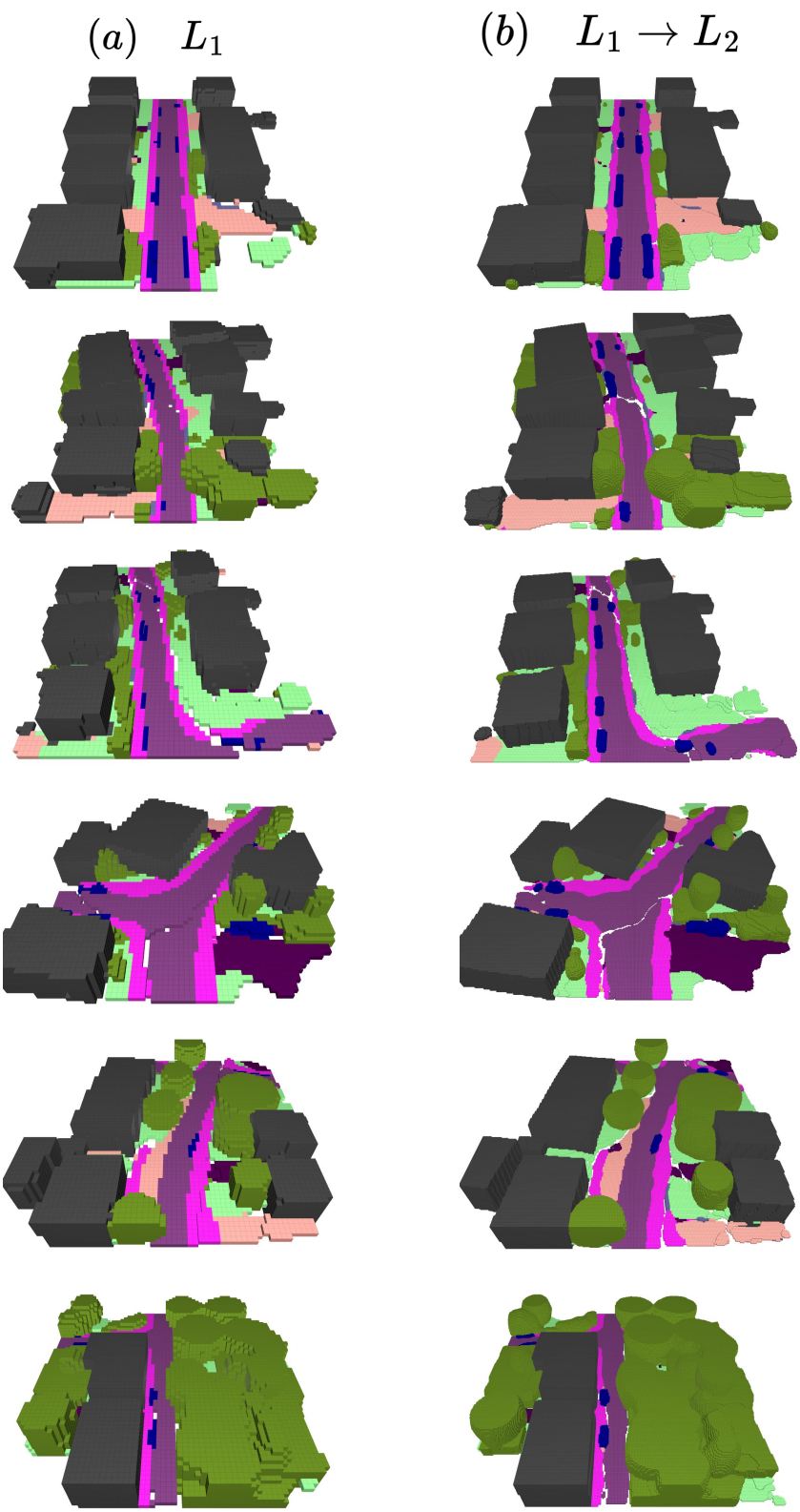


Figure 16. **XCube Hierarchical Scene Generation Results.** Unconditional 3D semantic scenes generated by XCube [20] across successive hierarchy levels. Scene details improve across levels, and the overall structure appears realistic, but irregular object shapes (e.g. vehicles) and occasional road-geometry discontinuities remain visible.

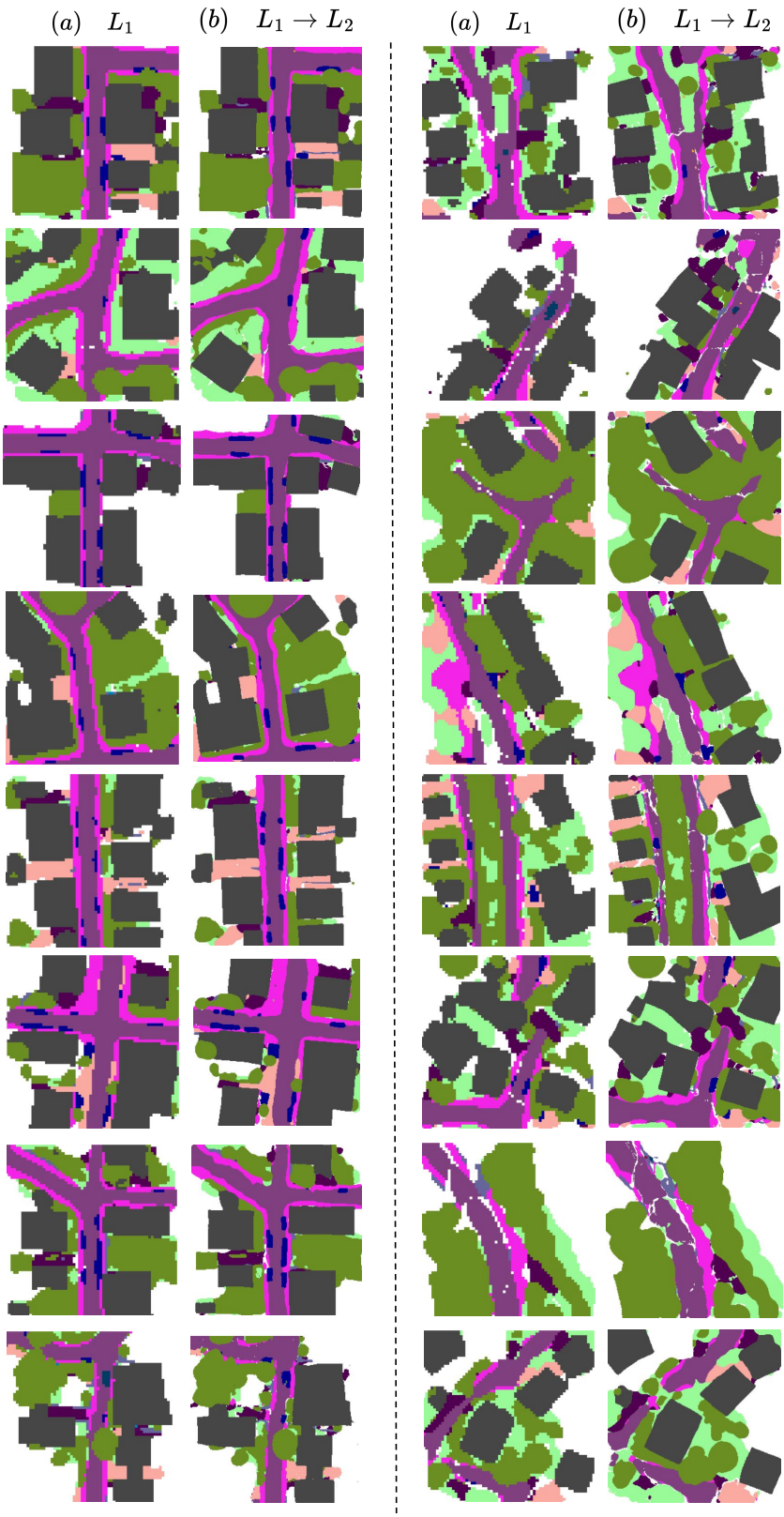


Figure 17. **Hierarchical BEV Renderings of XCube-generated Scenes.** Each column pair shows results across successive hierarchy levels. Examples on the left demonstrate high-quality generations, while those on the right exhibit geometric distortions and semantic inconsistencies.

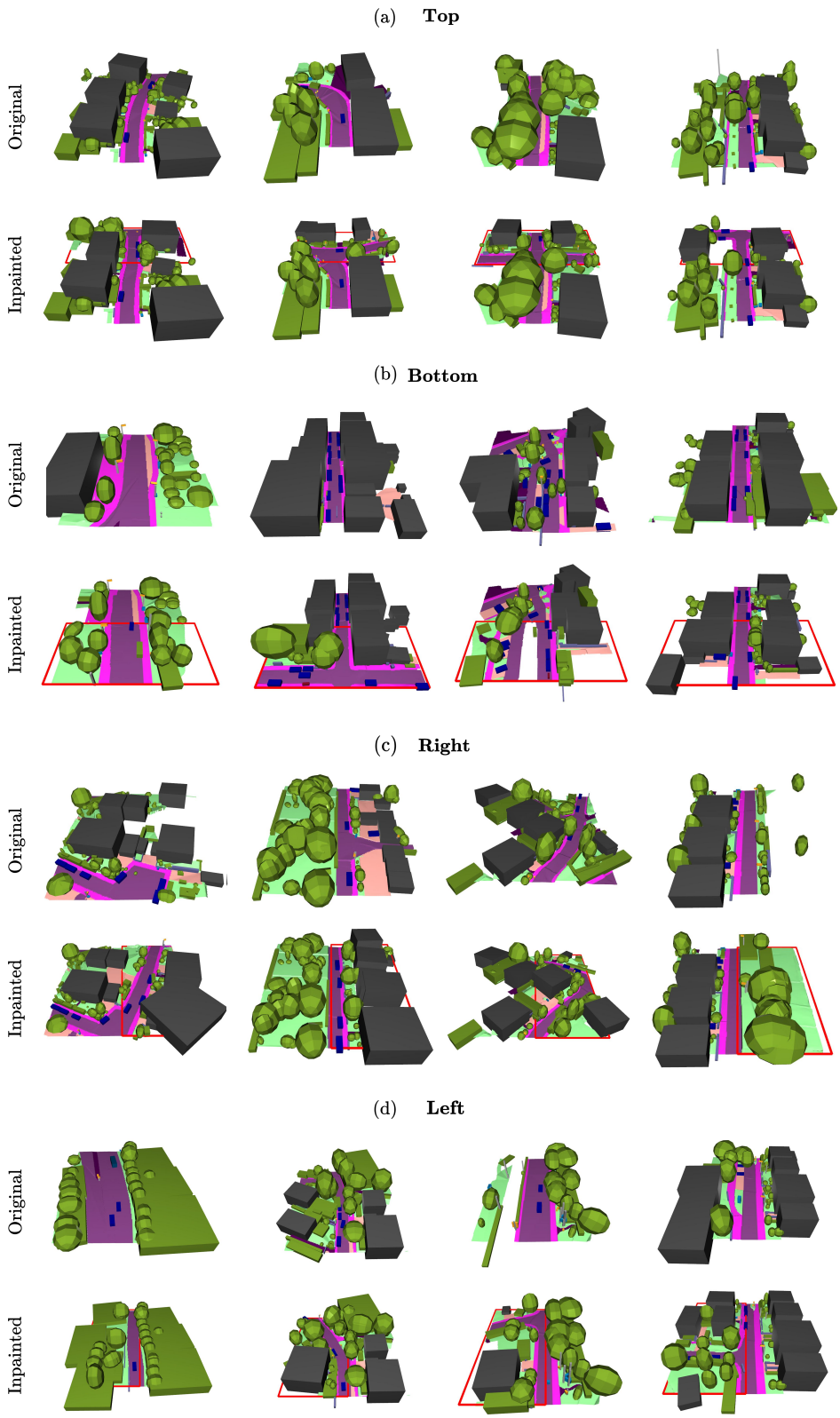


Figure 18. **PrITTI Inpainting Results.** Inpainting applied to (a) top, (b) bottom, (c) right, and (d) left regions of an input scene. PrITTI produces localized edits that blend seamlessly with the surrounding geometry and semantics, yielding consistent and plausible completions.

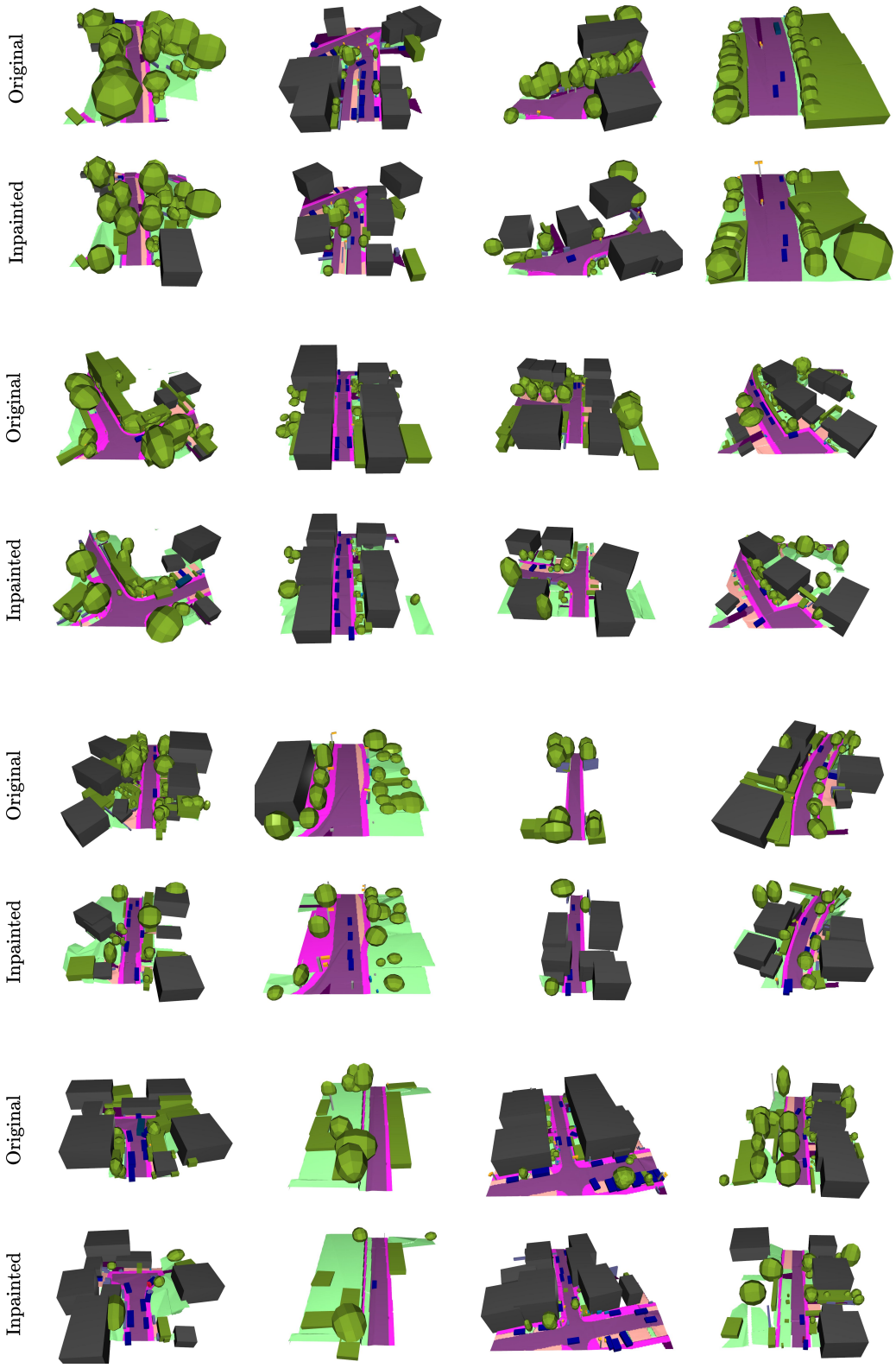


Figure 19. **PrITTI Ground-to-Object Inpainting Results.** Leveraging our disentangled joint latent structure, we can inpaint object primitives while keeping the ground layout fixed. The results show that the ground geometry is faithfully preserved from the original scene, while newly generated objects are placed in semantically appropriate locations.

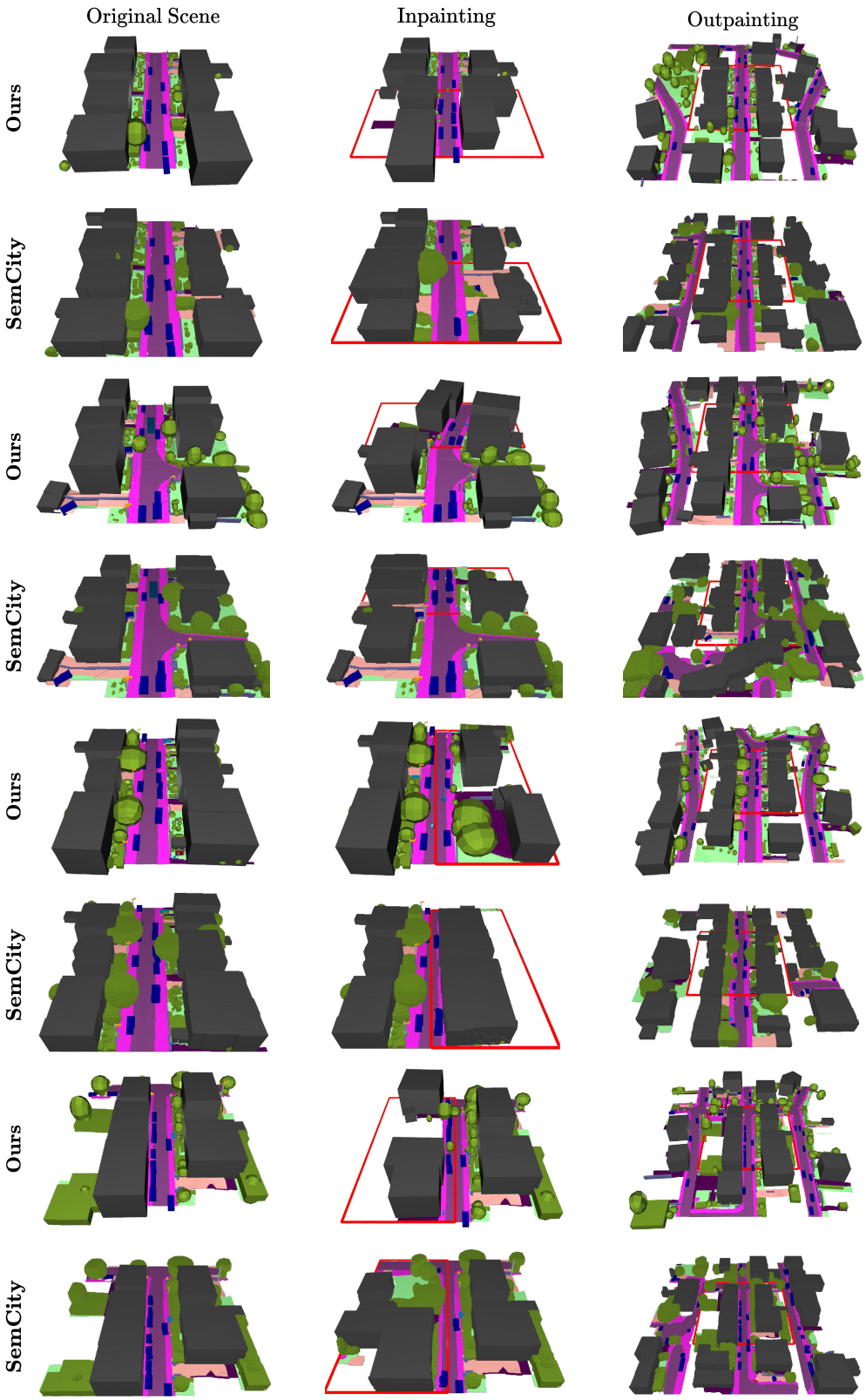


Figure 20. **Comparison of scene inpainting and outpainting results** on identical test scenes using our method and SemCity [13]. PrITTI (Ours) yields coherent and semantically consistent edits and extensions, while SemCity often produces fragmented or contextually inconsistent structures, particularly during outpainting.

Figure 21. **PrITTI Outpainting Results.** Outpainting examples produced by our method showing that it can expand a scene beyond its spatial extent while preserving ground layout continuity and generating new objects that integrate naturally with the surrounding context.

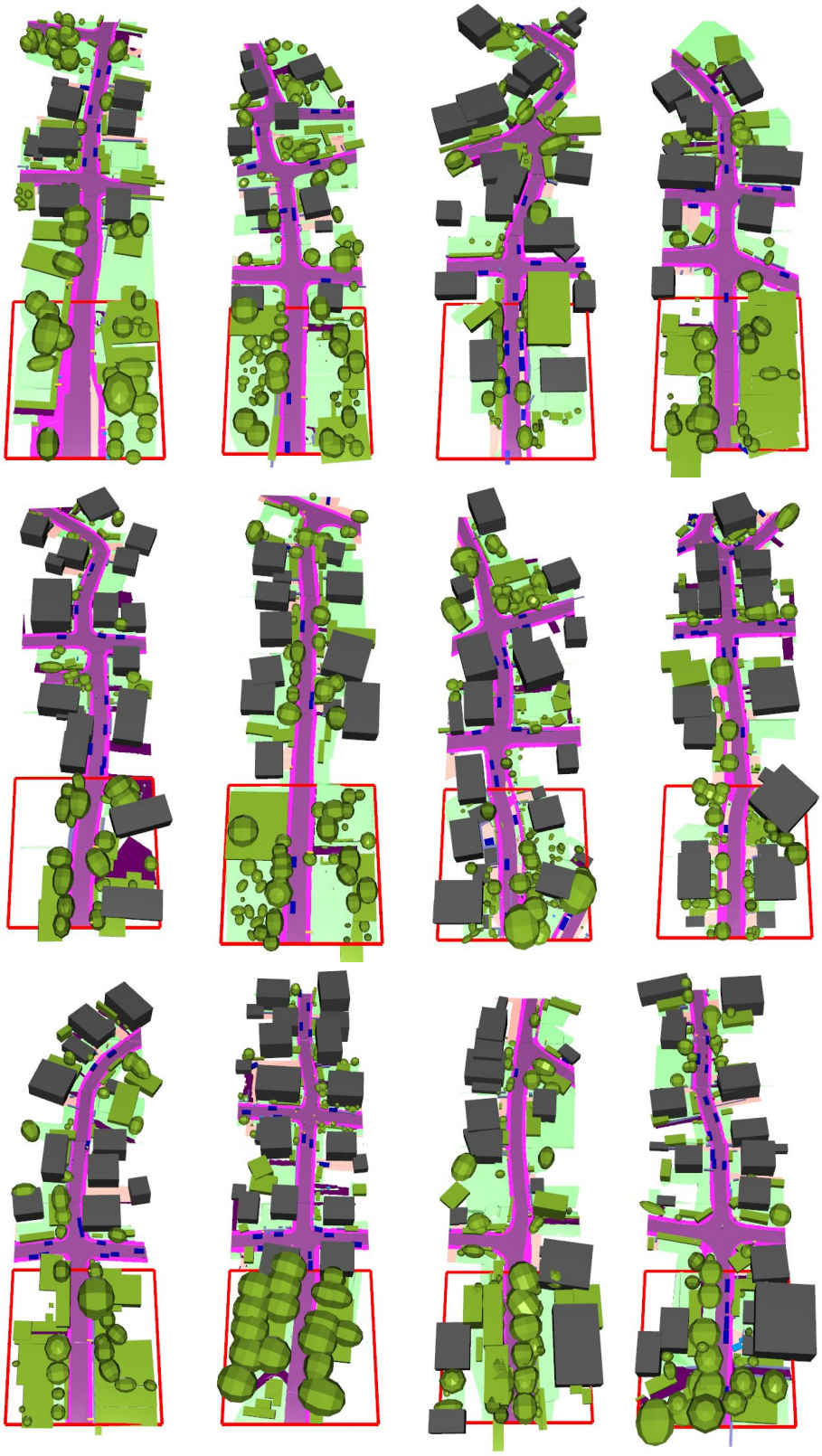


Figure 22. **PrITTI Large-scale Scene Extrapolation Results.** Starting from an initially generated block (outlined in red), our method produces extended layouts that remain globally coherent and semantically consistent, with smooth structural transitions across blocks.

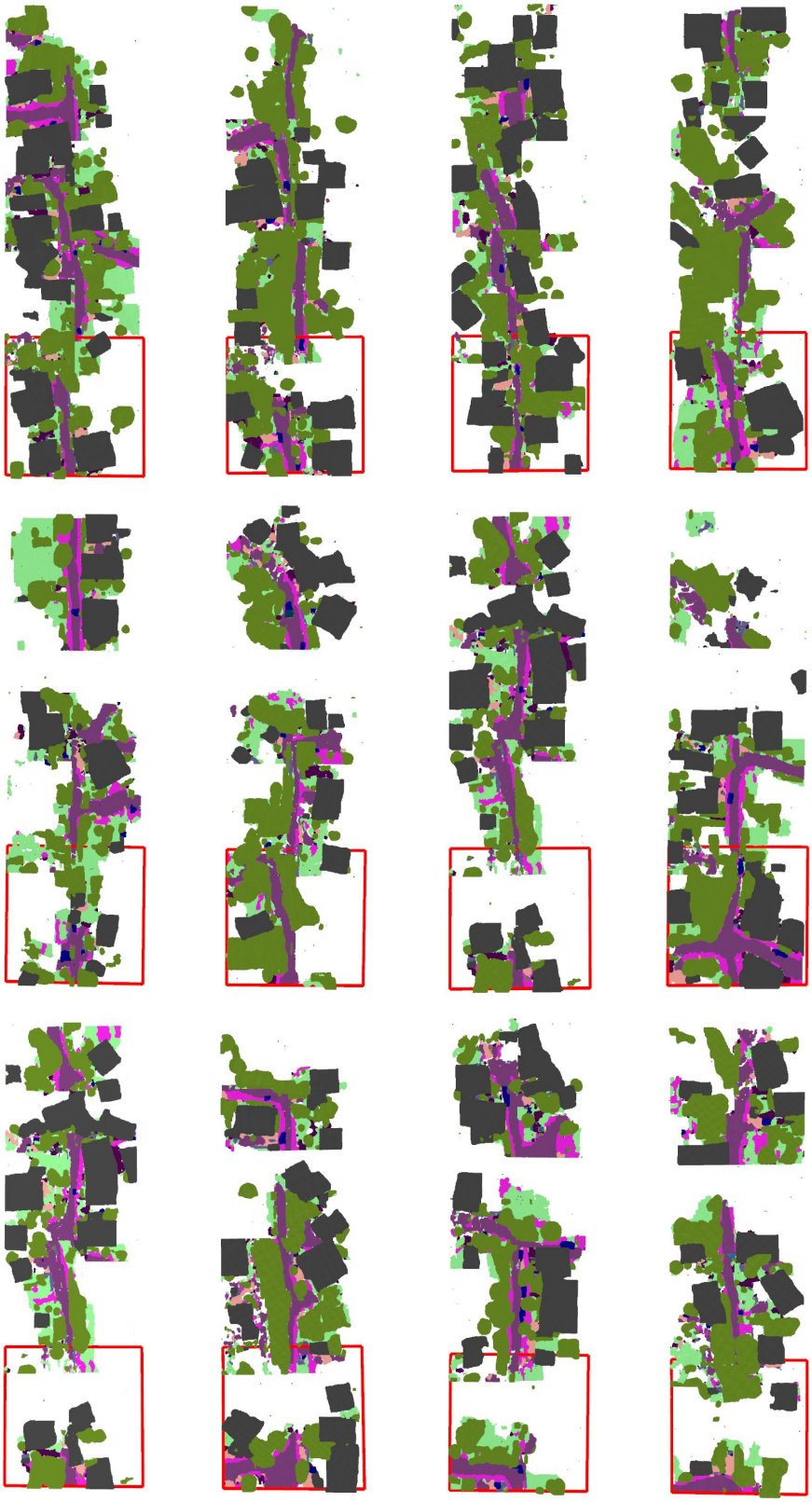


Figure 23. **PDD Large-scale Scene Extrapolation Results.** Starting from an initially generated block (outlined in red), PDD progressively extends the scene upward. The extrapolated regions exhibit broken road connectivity, misplaced structures, and inconsistent semantic transitions, leading to fragmented large-scale layouts.

Figure 24. **PrITTI Object Editing Results.** Enabled by our instance-level primitive representation, PrITTI naturally supports a wide range of individual-object manipulations. Columns (b-e) show dropout, rotation, scaling, and translation applied to the same randomly selected vehicle primitives from (a), highlighted in white.

Figure 25. **PrITTI Photo-realistic Street View Synthesis Results.** Each pair shows a 2D semantic rendering (left) and the corresponding photo-realistic image (right), generated with ControlNet [25]. Despite the coarse geometry of our primitives, the synthesized scenes exhibit diverse object appearances that go beyond cuboids or ellipsoids while remaining consistent with the input semantics.

References

[1] 3d karton city model. https://www.turbosquid.com/3d-models/3d-karton-city-2-model-1196110. 7

[2] Jens Behley, Martin Garbade, Andres Milioto, Jan Quenzel, Sven Behnke, Cyrill Stachniss, and Jurgen Gall. Semantickitti: A dataset for semantic scene understanding of lidar sequences. In *Proc. of the IEEE International Conf. on Computer Vision (ICCV)*, 2019. 7

[3] Holger Caesar, Varun Bankiti, Alex H Lang, Sourabh Vora, Venice Erin Liong, Qiang Xu, Anush Krishnan, Yu Pan, Giancarlo Baldan, and Oscar Beijbom. nuscenes: A multimodal dataset for autonomous driving. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2020. 10

[4] Holger Caesar, Juraj Kabzan, Kok Seang Tan, Whye Kit Fong, Eric M. Wolff, Alex H. Lang, Luke Fletcher, Oscar Beijbom, and Sammy Omari. nuplan: A closed-loop ml-based planning benchmark for autonomous vehicles. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR) Workshops*, 2021. 10

[5] Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko. End-to-end object detection with transformers. In *Proc. of the European Conf. on Computer Vision (ECCV)*, 2020. 3

[6] A. Cholesky. Sur la résolution numérique des systèmes d'équations linéaires. *Bulletin Géodésique*, 1924. 2

[7] Prafulla Dhariwal and Alexander Nichol. Diffusion models beat gans on image synthesis. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2021. 9

[8] Jonathan Ho and Tim Salimans. Classifier-free diffusion guidance. *arXiv.org*, 2207.12598, 2022. 6

[9] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020. 5

[10] Diederik P. Kingma and Max Welling. Auto-encoding variational bayes. *Proc. of the International Conf. on Learning Representations (ICLR)*, 2014. 4

[11] Harold W Kuhn. The hungarian method for the assignment problem. 1955. 3

[12] Tuomas Kynkäänniemi, Tero Karras, Samuli Laine, Jaakko Lehtinen, and Timo Aila. Improved precision and recall metric for assessing generative models. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019. 9

[13] Jumin Lee, Sebin Lee, Changho Jo, Woobin Im, Juhyeong Seon, and Sung-Eui Yoon. Semcity: Semantic scene generation with triplane diffusion. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2024. 7, 11, 18, 26

[14] Yiyi Liao, Jun Xie, and Andreas Geiger. Kitti-360: A novel dataset and benchmarks for urban scene understanding in 2d and 3d. *IEEE Trans. on Pattern Analysis and Machine Intelligence (PAMI)*, 2022. 1, 11

[15] Yuheng Liu, Xinke Li, Xueting Li, Lu Qi, Chongshou Li, and Ming-Hsuan Yang. Pyramid diffusion for fine 3d large scene generation. In *Proc. of the European Conf. on Computer Vision (ECCV)*, 2024. 6, 7, 11, 20

[16] Fan Lu, Kwan-Yee Lin, Yan Xu, Hongsheng Li, Guang Chen, and Changjun Jiang. Urban architect: Steerable 3d urban scene generation with layout prior. *arXiv.org*, 2404.06780, 2024. 9

[17] Jack Lu, Kelvin Wong, Chris Zhang, Simon Suo, and Raquel Urtasun. Scenecontrol: Diffusion for controllable traffic scene generation. In *Proc. IEEE International Conf. on Robotics and Automation (ICRA)*, 2024. 11

[18] Andreas Lugmayr, Martin Danelljan, Andres Romero, Fisher Yu, Radu Timofte, and Luc Van Gool. Repaint: Inpainting using denoising diffusion probabilistic models. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2022. 5

[19] William Peebles and Saining Xie. Scalable diffusion models with transformers. In *Proc. of the IEEE International Conf. on Computer Vision (ICCV)*, 2023. 5, 9

[20] Xuanchi Ren, Jiahui Huang, Xiaohui Zeng, Ken Museth, Sanja Fidler, and Francis Williams. Xcube: Large-scale 3d generative modeling using sparse voxel hierarchies. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2024. 7, 11, 22

[21] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2022. 5

[22] Pei Sun, Henrik Kretzschmar, Xerxes Dotiwalla, Aurelien Chouard, Vijaysai Patnaik, Paul Tsui, James Guo, Yin Zhou, Yuning Chai, Benjamin Caine, Vijay Vasudevan, Wei Han, Jiquan Ngiam, Hang Zhao, Aleksei Timofeev, Scott Ettinger, Maxim Krivokon, Amy Gao, Aditya Joshi, Yu Zhang, Jonathon Shlens, Zhifeng Chen, and Dragomir Anguelov. Scalability in perception for autonomous driving: Waymo open dataset. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2020. 7

[23] Benjamin Wilson, William Qi, Tanmay Agarwal, John Lambert, Jagjeet Singh, Siddhesh Khandelwal, Bowen Pan, Ratnesh Kumar, Andrew Hartnett, Jhony Kaesemodel Pontes, Deva Ramanan, Peter Carr, and James Hays. Argoverse 2: Next generation datasets for self-driving perception and forecasting. In *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*, 2021. 10, 11, 13

[24] Joey Wilson, Jingyu Song, Yuewei Fu, Arthur Zhang, Andrew Capodieci, Paramsothy Jayakumar, Kira Barton, and Maani Ghaffari. Motionsc: Data set and network for real-time semantic mapping in dynamic environments. *IEEE Robotics and Automation Letters (RA-L)*, 2022. 7

[25] Lvmin Zhang, Anyi Rao, and Maneesh Agrawala. Adding conditional control to text-to-image diffusion models. In *Proc. of the IEEE International Conf. on Computer Vision (ICCV)*, 2023. 9, 31

[26] Haowen Zheng and Yanyan Liang. Sseditor: Controllable mask-to-scene generation with diffusion model. *arXiv.org*, 2411.12290, 2024. 11