AETREE: AREAL SPATIAL DATA GENERATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Areal spatial data represent not only geographical locations but also sizes and shapes of physical objects such as buildings in a city. Data-driven generation of such vector-format data requires an effective representation. Inspired by the hierarchical nature of such spatial data, we propose AETree, a tree-based deep auto-encoder network. Unlike common strategies that either treat the data as an unordered set or sort them into a sequence, we preprocess the data into a binary tree via hierarchical clustering. Then a tree encoder learns to extract and merge spatial information from bottom-up iteratively. The resulting global representation is reversely decoded for reconstruction or generation. Experiments on large scale 2D/3D building datasets of both New York and Zurich showed superior performance of AETree than either set-based or sequential auto-regressive deep models.

1 INTRODUCTION

Spatial data describes the geometry information of physical entities. We term those with areas/volumes as *areal spatial data*, recording both locations, sizes, and shapes in vector format, for objects such as buildings in a city[1]. As image/video generation evolves rapidly, similar data-driven generation for areal spatial data becomes more appealing due to 1) the abundance of such real-world data, and 2) the frequent use of such data in various applications. For example, urban planners and architects heavily rely on spatial simulations to structure their designs; game makers use spatial data generation tools to automatically create new virtual environments; and more recently, there is a surging demand from the autonomous driving industry to conduct road testing in simulated environments with novel maps and scenarios; all of these need areal spatial data generation.

While deep generative models are successful for various data modalities including language, audio, image, video, and even point clouds, several difficulties keep deep generation less explored for

[1]Roads are ignored in this term since they are often better modeled as polylines.

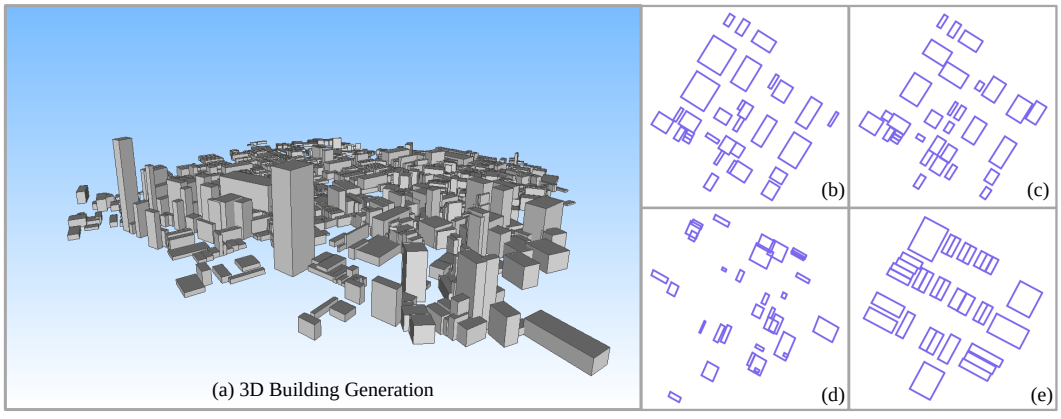

Figure 1: Example 3D (a) or 2D (b-e) building generation by AETree trained on New York City (NYC) dataset. Given a ground truth set of 2D building footprints in NYC (b), the AETree reconstruction result is shown in (c). The 2D generation results of SketchRNN (Ha & Eck, 2017) and AETree are shown in (d) and (e) respectively.

areal spatial data despite of their similarity to point clouds. First, they form a set of more complex geometrically parameterized objects with irregular layouts. Second, such a complex object itself may consists of a set of simpler objects. Third, these objects usually live on a high dimensional complex data manifold than simple points in point clouds. For example, a building of the second Level of Detail (LOD2) in CityGML format (Gröger et al., 2012) is recorded as a set of 3D polygons, each of which contains a variable number of 3D vertices. Although *this paper focuses on the simpler LOD1 where a building is just a cuboid*, generating a set of such cuiboids with resembling layouts as in the training dataset is *still considerably more difficult than point clouds*, as shown in Figure 1(d).

There exist two major strategies, set-based or sequence-based, to handle such a set of geometric objects. Set-based methods extract local features from each object independently, and aggregate them into a global representation (Qi et al., 2017), based on which generation can be done via direct multi-layer perceptron (MLP) (Achlioptas et al., 2018), deep parametric surfaces (Yang et al., 2018; Groueix et al., 2018), or deep implicit functions (Park et al., 2019). Sequence-based methods first manually sort those objects into a sequence which is then processed by auto-regressive recurrent neural networks (RNNs) (Ha & Eck, 2017; Chu et al., 2019) or attention-augmented graph neural networks (GNNs) (Nash et al., 2020). However, the set-based strategy cannot efficiently capture objects' local covariation, while the sequence-based one suffers from the loss of spatial neighborhood information in the sorted sequence.

The contribution of this paper is an alternative strategy based on *spatial hierarchy* as a more effective structure than set/sequence for representing areal spatial data. Our method AETree is based on this strategy and is shown to achieve superior reconstruction and generation performance than methods using the other two strategies in two large scale real-world building datasets.

2 Related work

Procedural modeling such as L-systems create geometric structures based on handcrafted production rules (Merrell et al., 2010; Vanegas et al., 2012b; Yang et al., 2013; Demir et al., 2014). Recently, inverse procedural modeling starts to learn the rules from data aided by deep nets (Vanegas et al., 2012a; Ritchie et al., 2016; Nishida et al., 2016; Guo et al., 2020). Differently, AETree can be seen as a *continuous L-system implemented as a deep net* and enables end-to-end learning from scratch.

Deep geometric data modeling has gained popularity in the last lustrum since it enables the generation of complex geometric structures (vertices/lines/surfaces) with less human input. To generate human sketch drawings, Ha & Eck (2017) proposed SketchRNN, an RNN model with a VAE structure to sequentially produce sketch strokes. Nash et al. (2020) proposed PolyGen to generate 3D polygon meshes with an autoregressive transformer model. House-GAN by Nauata et al. (2020) is an indoor layout generator enabled with GAN framework. To model city-level road layouts, Chu et al. (2019) proposed Neural Turtle Graphics (NTG), an encoder-decoder RNN model to generate roads sequentially. Zhang et al. (2020) further proposed a convolutional message passing neural network for supervised architecture reconstruction from images. Distinctively, AETree explores data's spatial hierarchy instead of using a sequence.

Tree-structured neural networks have long been explored for natural language processing tasks, such as as sentence parsing (Goller & Kuchler, 1996; Socher et al., 2011), representation learning (Tai et al., 2015; Li et al., 2015), and program generation (Chen et al., 2018). More recently, Roy et al. (2020) proposed Tree-CNN, a model to grow neural networks during incremental learning. On the geometric data, researchers have explored tree networks for 3D point cloud modeling. Klokov & Lempitsky (2017) proposed KD-Net that encodes point cloud features hierarchically with a KD-tree, and used it for classification and segmentation. Gadelha et al. (2018) further extended it to a multi-resolution tree networks for more efficient point cloud processing. However, both works focused on discriminative tasks, and the proposed models are not suitable for data generation.

3 AETree

Our goal is to discover the hierarchical structures in the areal spatial data (Section 3.1), and then apply a custom tree structured neural networks for encoding and generation (Section 3.2). Figure 2 gives an overview of our method by taking 2D box data as an example.

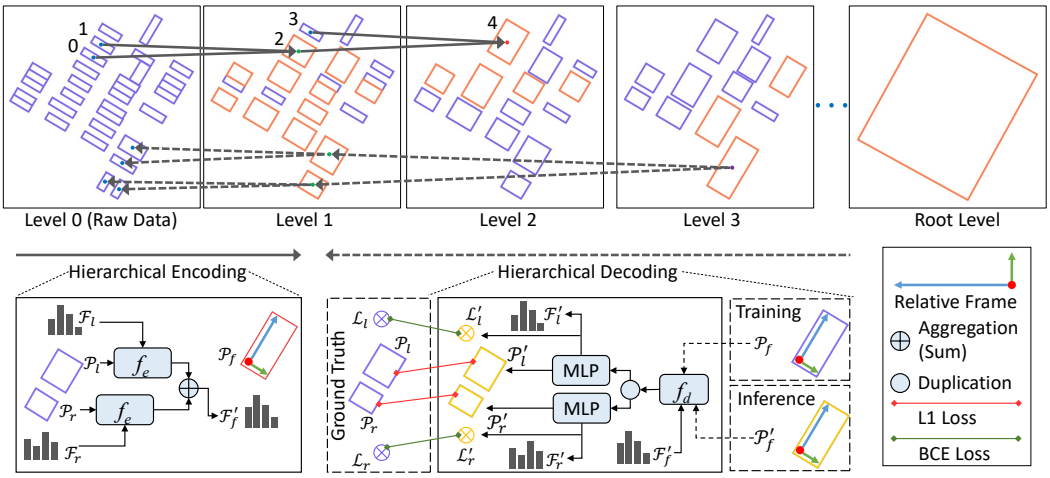

Figure 2: Illustration of the AETree model. The top row in the figure displays an example of pre-calculated tree structure from raw data level to root level. The orange boxes at the each level represent new boxes obtained by merging children boxes form the last level (for example, box 2 in level 1 is generated based on box 0 and 1 from level 0). The bottom row presents our encoding and decoding modules. Based on the tree structure, we first hierarchically encode children nodes to acquire the features of their father nodes until getting the root node features. Then starting from the root level, we hierarchically decode father nodes to children nodes and finally obtain the parameter of raw nodes.

3.1 Discovering Spatial Hierarchy in Data

Let us first consider a set of spatial data $\{\mathcal{P}_i\}\,|_{i=1,\dots,N}$, where $\mathcal{P}_i$ represents a single object instance in the set and N is the number of objects. As explained, we focus on 3D cuboids such as buildings, so $\mathcal{P} = (x, y, l, w, h, a) \in \mathbb{R}^6$, where x and y denote the center coordinates of a cuboid, and l, w, h and a denote the length, width, height and orientation angle of the cuboid.

To organize the data hierarchically in a binary tree $\mathcal{T}$, we apply hierarchical clustering (Johnson, 1967) by introducing $N-1$ intermediate nodes so that all the original objects stay on the leaf nodes. Concretely, the binary tree is built by recursively merging two closest nodes into a father node until we obtain a single root node.

The tree is homogeneous, so the intermediate nodes also represent 3D cuboids. For any intermediate node produced, its parameters x and y are obtained as the mean value of corresponding children nodes, and l, w, h and a are defined as the minimum bounding rectangle of its children nodes. Note that before feeding the data into our model, we choose to represent all node parameters *relative to their father nodes* (except the root node) as follows (subscript c/f means child/father),

$$x_c^r = \frac{x_c - x_f}{l_f}, \quad y_c^r = \frac{y_c - y_f}{w_f}, \quad l_c^r = \frac{l_c}{l_f}, \quad w_c^r = \frac{w_c}{w_f}, \quad h_c^r = \frac{h_c}{h_f}, \quad a_c^r = a_c - a_f.$$

We find this relative representation performs better in reconstruction, and it is only possible in this tree-based (instead of set/sequence) structure (more analysis in Appendix A.1).

The distance metric is also important for hierarchical clustering. The one we use is defined as:

$$D(i,j) = \lambda_1 D_{\text{center}}(i,j) + \lambda_2 D_{\text{area}}(i,j) + \lambda_3 D_{\text{shape}}(i,j) + \lambda_4 D_{\text{angle}}(i,j) + \lambda_5 D_{\text{merge}}(i,j),$$

where $D(i,j)$ represents the distance between cuboid i and j, λ represents the weight of each distance. Specifically, D_{center} measures the Euclidean distance between the center points of two cuboids; D_{area}, D_{shape} and D_{angle} separately measure the difference between the area, the aspect ratio, and the orientation of two cuboids; and D_{merge} measures the difference between the sum of the two cuboids area and their minimum bounding rectangle area. Mathematical definitions of these distances are detailed in Appendix A.2.

We present a simple example to explain our data structure. As shown in Figure 3, three trees are all produced based on four leaf nodes, but through different merging patterns. We introduce an index matrix I_l to store the merging rules at each level l, where in each row, the first two columns denote the indices of children nodes and the last column is the index of the father node. In this way, we can gather node features of all trees at the same level effectively, and put them in a mini-batch for training and inference.

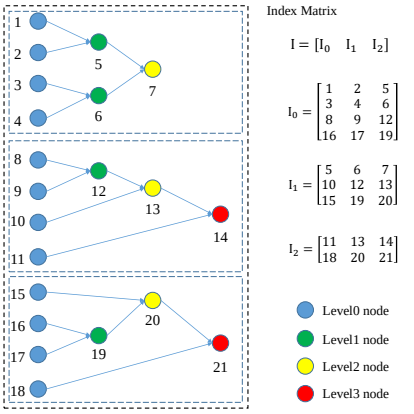

Figure 3: An example of our data structure. The blue nodes at level 0 store raw objects and we build a hierarchical clustering tree according to their pairwise distances. For efficient mini-batch training, we design an index matrix for each level storing the indexes of children and fathers.

3.2 Tree-shaped Auto-encoder Network

With the data constructed hierarchically, we can naturally develop an auto-encoding neural network with a tree shape to encode, decode and generate the cuboid sets. Our encoder learns a latent representation $\mathcal{F}_{root}$ of the root node by encoding each node from bottom to top. Conversely, our decoder decodes the root node from top to bottom and reconstructs the original data. Each node in the tree is represented by its geometric parameters $\mathcal{P}$ and a feature representation $\mathcal{F}$.

Encoding. To obtain the root node feature $\mathcal{F}_{root}$, the encoder encodes all intermediate nodes from bottom to top level by level. Given the the parameters and features of a left child and a right child, the feature of a father node is computed as:

$$\mathcal{F}'_f = f_e([\mathcal{P}_l, \mathcal{F}_l]) + f_e([\mathcal{P}_r, \mathcal{F}_r]), \tag{1}$$

where f_e represents an encoding function that encodes children parameters and features. In the experiments, we tried both MLP and LSTM cell as the f_e function. And it can be seen that our encoding function is symmetric, meaning the encoded father feature does not contain order information. Note that the parameters of father nodes $\mathcal{P}_f$ are pre-computed during data construction.

Decoding. The decoder aims to reconstruct the original data from the root features $\mathcal{F}_{root}$ produced by the encoder. At each level, we decode from a father node into two children nodes, which can be formulated as

$$[\mathcal{P}'_l, \mathcal{F}'_l, \mathcal{L}'_l, \mathcal{P}'_r, \mathcal{F}'_r, \mathcal{L}'_r] = f_d([\mathcal{P}'_f, \mathcal{F}'_f]) \tag{2}$$

where f_d denotes the decoding function, $\mathcal{P}'$, $\mathcal{F}'$ and $\mathcal{L}'$ indicate a node's decoded parameters, features, and indicator of being leaf nodes or not. We add this indicator judgement to determine whether the current node should be further decoded or not at inference time. Whenever all the nodes are identified as leaf nodes, the decoding process will stop.

During training, following the idea of teacher forcing, we use the ground-truth (pre-calculated) parameters $\mathcal{P}_f$ of the father node as the decoder input, which renders the model training faster and more efficient. L1 loss and Binary Cross-Entropy (BCE) loss are used to minimize the errors of predicted parameters and indicators, respectively.

Generation. To empower the model with data generation capability, we fit a Gaussian Mixture Model (GMM) on the root feature representation. Specifically, we obtain the root features of all the training data by passing into our encoder, and then estimate this distribution with a GMM. During the data generation process, we sample a new root feature $\mathcal{F}_g$ from the fitted GMM distribution, and a new data is generated going through our decoder.

4 Experiments

4.1 Dataset

We collected CityGML models of the New York City (NYC) from The New York City Department of Information Technology (2019). And then we extracted 955,120 individual building models,

where each is represented by polygon mesh, through parsing the raw CityGML data. Semantic information of buildings is preserved in polygon mesh through adding a class label to each surface in our dataset. There are 3 categories for building surfaces: ground surface, roof surface, and wall surface. Among these, we extracted the building footprint from ground surface and calculated the minimum bounding rectangle of each building footprint to get a single box. For each building, we perform the same process with its 32 neighbor buildings and consider the obtained 32 boxes as one set. Moreover, we obtained the 3D cuboid set by adding the height of building to 2D box set. In this paper, we selected 45,487 cuboid sets, which are generated based on all the buildings in Manhattan, as our dataset. Similarly, we obtained CityGML models of Zurich (Stadt Zurich, 2018), the largest city in Switzerland, and processed these models to obtain 52,225 cuboid sets. For each dataset, 70% of the data are chosen as training sets, 10% as valiation sets and 20% as test sets. Note that we conduct experiments both on 2D boxes and 3D cuboids datasets and we consider a 2D box as a simplified cuboid represented by (x, y, l, w, a), as described in Section 3.1.

4.2 Evaluation Metrics

To quantitatively evaluate areal spatial data generation, we first convert spatial data to point clouds (a cuboid is converted to the eight corner points) and then adopt three popular metrics (Achlioptas et al., 2018). Jensen-Shannon Divergence (**JSD**) measures the similarity of marginal distributions between the reference and generated sets. The distribution of data is calculated by counting the number of points in each discretized grid cell. Coverage (**COV**) measures the fraction of points in generated data that are matched to the corresponding closest neighbor points in the reference data. Minimum Matching Distance (**MMD**) measures the fidelity of generated set with respect to reference set by matching each generated point to the point in reference data with the minimum distance.

For COV and MMD, we only select Chamfer Distance (**CD**) to compute the distance between two point clouds. We leave out Earth Mover's Distance (**EMD**) as it requires the number of instances in two sets to be equal, which is not suitable for our generation evaluation. Moreover, compared to points, an unique aspect of box data is their spatial extents. Therefore, we introduce another metric, Overlapping Area Ratio (**OAR**), which measures the area ratio of overlapped to all objects.

4.3 Implementation details

As stated in Section 3.1, a comprehensive distance metric which integrates five distance metrics with different weights is defined for hierarchical clustering. After performing a series of comparison experiments, the weights λ_{1-5} are set to 5, 2, 0.1, 1 and 1 respectively. Likewise, the GMMs of 60 and 80 components both with full covariance are selected as generators for the NYC and the Zurich dataset, respectively.

The AETree model is implemented based on the Pytorch framework. We employ the ADAM optimizer with a learning rate of 0.001 and divide the learning rate by 2 every 400 steps. The batch size of tree data is set to 50 in our experiments. And for the JSD metric, the number of points are counted by discretizing the space into 28^3 voxels.

4.4 Results

4.4.1 Reconstruction

We first present the reconstruction results of our models. To show the effectiveness of different encoding and decoding functions, we employ the Multilayer Perceptron (MLP) and Long Short-Term Memory (LSTM) cell as basic function separately. Similarly, we transform spatial data to point clouds and conduct quantitative evaluation under CD, EMD, and OAR metrics. Note that the reason why we employ EMD as the reconstruction metric here is that the number of reconstructed and original spatial data are equal. Table 1 summarizes the comparison results of two models on NYC and Zurich Dataset. It can be seen that the LSTM cell module performs better than MLP as base function. So the following results of our model are based on tree neural network with LSTM cell if no special explanation is provided. Moreover, we plot the reconstruction results in Figure 4, which demonstrates the promising performance our model achieves. More visualization of reconstruction can be found in Appendix A.3.

Table 1: Quantitative comparisons of AETree with different base nets on NYC and Zurich dataset. ↑: the higher the better, ↓: the lower the better. The best values are highlighted in bold.

Methods	Dataset	CD(↓)	EMD(↓)	OAR(%, ↓)
AETree (MLP)	NYC	0.0079	0.1206	5.81
AETree (LSTMCell)	NYC	**0.0019**	**0.0417**	**0.57**
AETree (MLP)	Zurich	0.0077	0.1163	2.04
AETree (LSTMCell)	Zurich	**0.0027**	**0.0580**	**0.14**

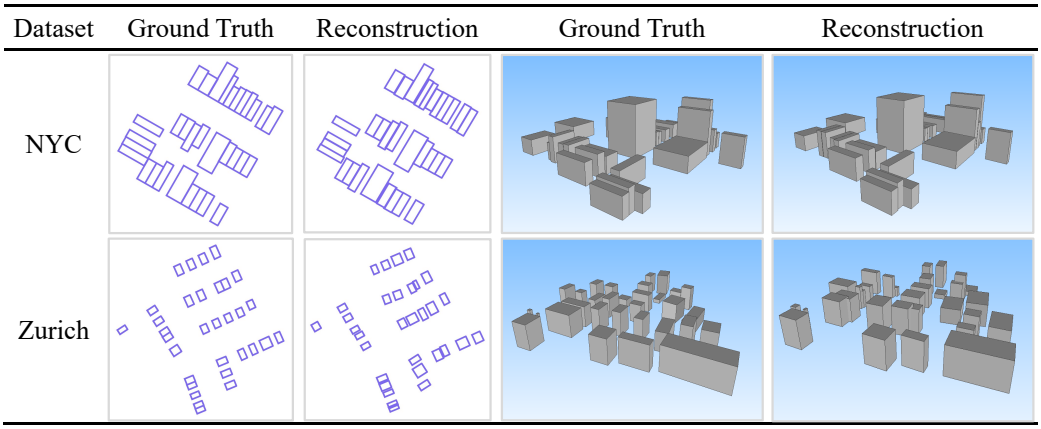

Figure 4: 2D and 3D building reconstruction results of AETree on NYC and Zurich dataset.

4.4.2 Latent Space Interpolation

Given latent representations of two box sets, we can obtain the intermediate box set by applying the decoder to the linear interpolation between these two latent spaces. Figure 5 shows the reconstructed box sets from the interpolated latent vectors. Interestingly, we produce a gradually varied sequence of box set from box set S to box set T, which demonstrates the smoothness of our latent space. Meanwhile, it can be found that our learned latent representations are generative, instead of being simply able to memorize the training sets.

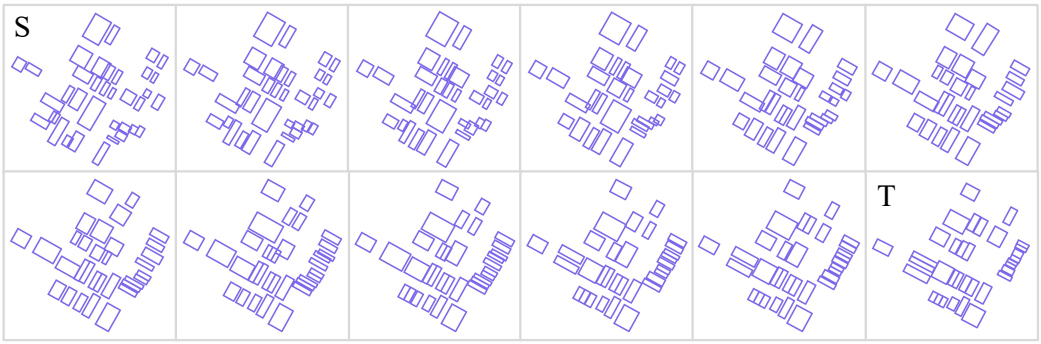

Figure 5: Latent space interpolation between two box sets, S and T, using AETree.

4.4.3 Generation

After acquiring the generative latent representations, we fit Gaussian Mixture Models (GMMs) and sample new latent representations, which are used as our decoder input to generate new box sets. We compare with three baseline methods:

SketchRNN-R2. SketchRNN is a generative model to generate sketch drawings (Ha & Eck, 2017). This model seems intuitively suitable to solve our problem. We convert each box set to a list of points as a sketch according to the input of SketchRNN. Specifically, for a box set with 32 boxes, the converted sketch consists of 128 points in 5D as in (Ha & Eck, 2017).

SketchRNN-R5. Based on vanilla SketchRNN, we explore replacing the parameter of a box (defined in Section 3.1) with x,y coordinates of a sketch. So we transform 32 boxes in a set to a sketch with 32 high dimension points, which incorporates 6 elements: (Δx, Δy, Δl, Δw, Δa, p). The first five elements are the offset parameters from the previous box. Different from Ha & Eck (2017), we use 1D to represent the binary state of the pen (at its end or not), since we assume that the pen draws 32 points in succession.

PointNet-MLP. In addition, we achieve a simple baseline model, which adopts PointNet (Qi et al., 2017) as the encoder by regarding a box set as a point cloud. By reference to the decoder of SketchRNN, we employ MLP to decode the latent representations to parameters for a probability distribution of points. Meanwhile, the loss function aims to maximize the log-likelihood of the generated probability distribution to explain the training data.

Table 2 summarizes the quantitative comparison generation results of the above three baseline methods and our models. It can be found that our model outperforms all baselines across four evaluation metrics. To intuitively show the superiority of our model, we randomly select some generation results of each model, as shown in Figure 6. We can find that the generations of SketchRNN and our model are more regular than the other two models, which coincide more with the city layout. By comparing the first two baseline models, we can see that the SketchRNN model is more suitable for the input with low-dimension coordinates instead of compact high-dimension representations. On the contrary, based on the input with box parameter representations, our model achieves better generative performance than the two SketchhRNN baseline methods. More 3D generation results of our model are shown in Appendix A.4.

In addition, the learnable parameters and floating point operations (FLOPs) of each method are presented in the last two columns of Table 2. It can be found that the number of parameters of all methods are very close but their FLOPs differs a lot. Though PointNet-MLP model demonstrates the lowest complexity both on the number of parameters and FLOPs, its generation results is unsatisfying on the four evaluation metrics. On the other hand, the generation results of SketchRNN-R2 is acceptable, yet it requires a higher computational cost. By comparison, our model achieves the best generation performance with a relatively lower complexity computation.

Table 2: Quantitative comparisons of generation performance with various baseline methods. The first four columns represent the results of models under four generation evaluation metrics and the last two columns measure the complexity of models.

Methods	JSD(↓)	COV(%, ↑)	MMD(↓)	OAR(%, ↓)	#params	FLOPs/sample
SketchRNN-R2	0.0089	33.62	0.0050	1.83	2.19M	243.13M
SketchRNN-R5	0.0101	28.76	0.0047	95.41	2.37M	402.46M
PointNet-MLP	0.0417	4.60	0.0219	87.47	**1.84M**	**3.67M**
AETree	**0.0033**	**39.53**	**0.0044**	**1.66**	2.91M	31.86M

4.5 Hyperparameter Analysis

Distance metrics. As described in Section 3.1, we defined a comprehensive distance metric, a weighted sum of five distance metrics. The weight λ of each distance is determined by comparison experiments, and the values of weight in our experiments are listed in the first row of Table 3. To explore the effect of each distance, we multiply 5 by the value of λ_i respectively. As presented in Table 3, there is no big difference between these results except the change of λ_5, which means that the tree structure may be less effective if we put too much weight on minimizing D_{merge}.

Table 3: The reconstruction comparisons of different distance metrics for hierarchical clustering.

$\lambda 1$	$\lambda 2$	$\lambda 3$	$\lambda 4$	$\lambda 5$	CD(↓)	EMD(↓)	OAR(%, ↓)
5	2	0.1	1	1	**0.0019**	**0.0417**	0.57
25	-	-	-	-	0.0021	0.0449	0.45
-	10	-	-	-	0.0019	0.0419	0.50
-	-	0.5	-	-	0.0022	0.0456	0.93
-	-	-	5	-	0.0020	0.0421	**0.41**
-	-	-	-	5	0.0042	0.0760	5.65

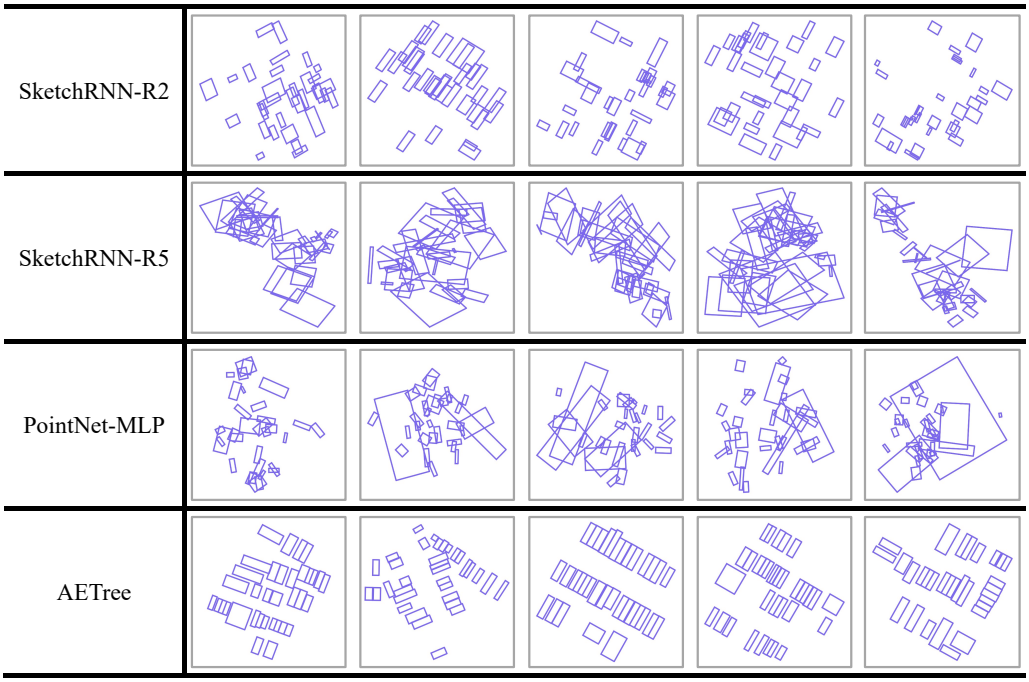

Figure 6: Unconditional 2D building generation results of the models trained on NYC dataset.

GMM parameters. To determine the optimal number and covariance type of Gaussian components for the GMM, we conduct a grid search using the JSD criterion. As shown in Table 4, the GMM of 60 components with full covariance obtains the optimal JSD value, which is thereby chosen for our generation experiments.

Table 4: The generation results of GMMs with a varying number and covariance type of Gaussian components under JSD metrics. Each GMM is trained on the latent space learned by AETree.

cov. types \ #components	10	20	30	40	50	60	70	80	90	100
full	0.0044	0.0039	0.0037	0.0036	0.0037	**0.0033**	0.0034	0.0034	0.0035	0.0034
diag	0.0176	0.014	0.0177	0.0118	0.0127	0.0171	0.0105	0.0118	0.0133	0.0255
tied	0.0057	0.0042	0.0045	0.0040	0.0043	0.0037	0.0043	0.004	0.0039	0.0043
spherical	0.0057	0.0060	0.0060	0.0059	0.0059	0.0057	0.0063	0.0061	0.0061	0.0062

5 Conclusions

In this work, we propose a tree-based deep auto-encoder network, AETree, to achieve reconstruction and generation of areal spatial data. Different from previous work that would treat areal spatial data as a set or a sequence, we employ a binary tree structure, making our model a continuous L-system implemented as a deep net and enables end-to-end learning from scratch. Experiments in two large scale real-world building datasets demonstrate the effectiveness of this hierarchical structure to take advantage of the spatial hierarchy that can be efficiently discovered by hierarchical clustering via data preprocessing. Although currently our model generates results that may not look as impressive as those from the inverse procedural generation, we believe our model offers a promising alternative for generic 2D/3D geometric content generation that can more efficiently benefit directly from a large amount of real-world spatial data, similar to the motivation for deep image/video generation.

In our future work, we plan to extend AETree to generate complex polygon/polyhedron sets where a polygon/polyhedron can be treated as an additional feature inside a box/cuboid. We will also make the generator condition on maps or human inputs, therefore enabling more possibilities for downstream applications.

REFERENCES

Panos Achlioptas, Olga Diamanti, Ioannis Mitliagkas, and Leonidas Guibas. Learning representations and generative models for 3d point clouds. In International conference on machine learning, pp. 40–49. PMLR, 2018. 2, 5

Xinyun Chen, Chang Liu, and Dawn Song. Tree-to-tree neural networks for program translation. In Advances in neural information processing systems, pp. 2547–2557, 2018. 2

Hang Chu, Daiqing Li, David Acuna, Amlan Kar, Maria Shugrina, Xinkai Wei, Ming-Yu Liu, Antonio Torralba, and Sanja Fidler. Neural turtle graphics for modeling city road layouts. In Proceedings of the IEEE International Conference on Computer Vision, pp. 4522–4530, 2019. 2

Ilke Demir, Daniel G Aliaga, and Bedrich Benes. Proceduralization of buildings at city scale. In 2014 2nd International Conference on 3D Vision, volume 1, pp. 456–463. IEEE, 2014. 2

Matheus Gadelha, Rui Wang, and Subhransu Maji. Multiresolution tree networks for 3d point cloud processing. In Proceedings of the European Conference on Computer Vision (ECCV), pp. 103–118, 2018. 2

Christoph Goller and Andreas Kuchler. Learning task-dependent distributed representations by backpropagation through structure. In Proceedings of International Conference on Neural Networks (ICNN'96), volume 1, pp. 347–352. IEEE, 1996. 2

Gerhard Gröger, Thomas H Kolbe, Claus Nagel, and Karl-Heinz Häfele. Ogc city geography markup language (citygml) encoding standard. 2012. 2

Thibault Groueix, Matthew Fisher, Vladimir G Kim, Bryan C Russell, and Mathieu Aubry. A papier-mâché approach to learning 3d surface generation. In Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 216–224, 2018. 2

Jianwei Guo, Haiyong Jiang, Bedrich Benes, Oliver Deussen, Xiaopeng Zhang, Dani Lischinski, and Hui Huang. Inverse procedural modeling of branching structures by inferring l-systems. ACM Transactions on Graphics (TOG), 39(5):1–13, 2020. 2

David Ha and Douglas Eck. A neural representation of sketch drawings. arXiv preprint arXiv:1704.03477, 2017. 1, 2, 7

Stephen C Johnson. Hierarchical clustering schemes. Psychometrika, 32(3):241–254, 1967. 3

Roman Klokov and Victor Lempitsky. Escape from cells: Deep kd-networks for the recognition of 3d point cloud models. In Proceedings of the IEEE International Conference on Computer Vision, pp. 863–872, 2017. 2

Jiwei Li, Minh-Thang Luong, Dan Jurafsky, and Eudard Hovy. When are tree structures necessary for deep learning of representations? arXiv preprint arXiv:1503.00185, 2015. 2

Paul Merrell, Eric Schkufza, and Vladlen Koltun. Computer-generated residential building layouts. In ACM SIGGRAPH Asia 2010 papers, pp. 1–12. 2010. 2

Charlie Nash, Yaroslav Ganin, SM Eslami, and Peter W Battaglia. Polygen: An autoregressive generative model of 3d meshes. arXiv preprint arXiv:2002.10880, 2020. 2

Nelson Nauata, Kai-Hung Chang, Chin-Yi Cheng, Greg Mori, and Yasutaka Furukawa. House-gan: Relational generative adversarial networks for graph-constrained house layout generation. arXiv preprint arXiv:2003.06988, 2020. 2

Gen Nishida, Ignacio Garcia-Dorado, Daniel G Aliaga, Bedrich Benes, and Adrien Bousseau. Interactive sketching of urban procedural models. ACM Transactions on Graphics (TOG), 35(4): 1–11, 2016. 2

Jeong Joon Park, Peter Florence, Julian Straub, Richard Newcombe, and Steven Lovegrove. Deepsdf: Learning continuous signed distance functions for shape representation. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 165–174, 2019. 2

Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 652–660, 2017. 2, 7

Daniel Ritchie, Anna Thomas, Pat Hanrahan, and Noah Goodman. Neurally-guided procedural models: Amortized inference for procedural graphics programs using neural networks. In Advances in neural information processing systems, pp. 622–630, 2016. 2

Deboleena Roy, Priyadarshini Panda, and Kaushik Roy. Tree-cnn: a hierarchical deep convolutional neural network for incremental learning. Neural Networks, 121:148–160, 2020. 2

Richard Socher, Cliff C Lin, Chris Manning, and Andrew Y Ng. Parsing natural scenes and natural language with recursive neural networks. In Proceedings of the 28th international conference on machine learning (ICML-11), pp. 129–136, 2011. 2

Stadt Zurich. Zurich 3-d building model. https://www.stadt-zuerich.ch/ted/de/index/geoz/geodaten_u_plaene/3d_stadtmodell.html, 2018. 5

Kai Sheng Tai, Richard Socher, and Christopher D Manning. Improved semantic representations from tree-structured long short-term memory networks. arXiv preprint arXiv:1503.00075, 2015. 2

The New York City Department of Information Technology. NYC 3-d building model. https://www1.nyc.gov/site/doitt/initiatives/3d-building.page, 2019. 4

Carlos A Vanegas, Ignacio Garcia-Dorado, Daniel G Aliaga, Bedrich Benes, and Paul Waddell. Inverse design of urban procedural models. ACM Transactions on Graphics (TOG), 31(6):1–11, 2012a. 2

Carlos A Vanegas, Tom Kelly, Basil Weber, Jan Halatsch, Daniel G Aliaga, and Pascal Müller. Procedural generation of parcels in urban modeling. In Computer graphics forum, volume 31, pp. 681–690. Wiley Online Library, 2012b. 2

Yaoqing Yang, Chen Feng, Yiru Shen, and Dong Tian. Foldingnet: Point cloud auto-encoder via deep grid deformation. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 206–215, 2018. 2

Yong-Liang Yang, Jun Wang, Etienne Vouga, and Peter Wonka. Urban pattern: Layout design by hierarchical domain splitting. ACM Transactions on Graphics (TOG), 32(6):1–12, 2013. 2

Fuyang Zhang, Nelson Nauata, and Yasutaka Furukawa. Conv-mpn: Convolutional message passing neural network for structured outdoor architecture reconstruction. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 2798–2807, 2020. 2

A APPENDIX

A.1 DISCUSSION ABOUT RELATIVE REPRESENTATION

As described in Section 3.1, we preprocess all nodes' parameters relative to their father nodes. To verify the effectiveness of this relative representation, we conduct a comparison experiment between different ways of representation. In Figure 7, we plot the reconstruction results of models with relative and absolute representation on NYC dataset. It can be found that relative representation keep the spatial relations among objects better than absolute representation, which is thus employed in our tree-based structure data.

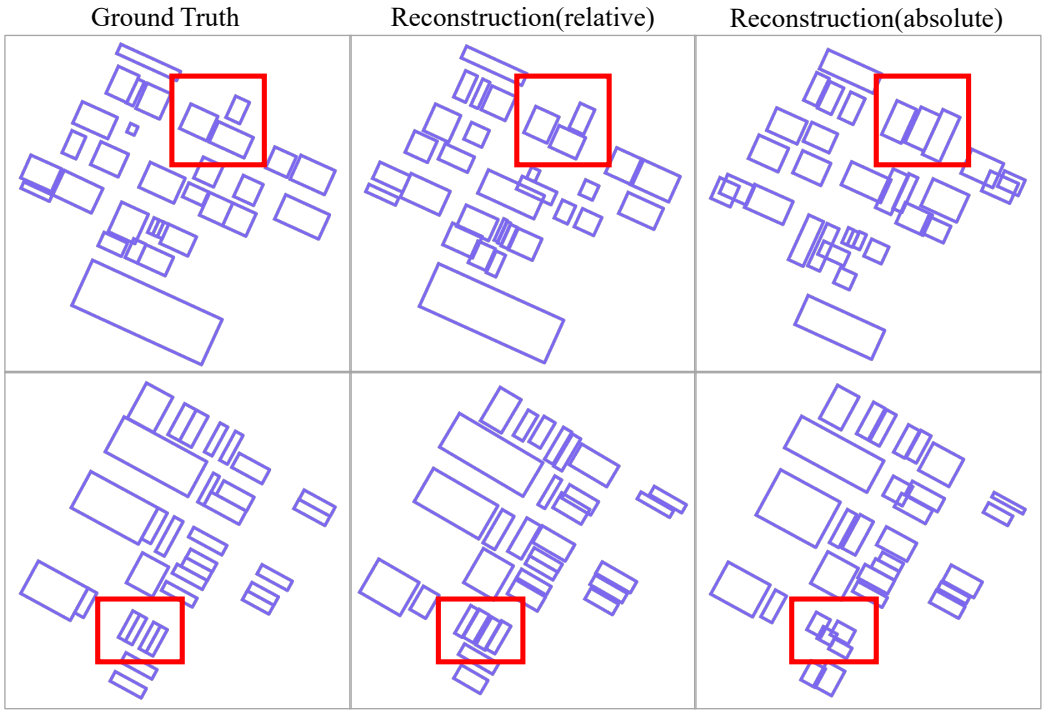

Figure 7: Reconstruction results of models with the relative and absolute representation of spatial data. The red boxes highlight the most obvious difference between results.

A.2 DISTANCE METRIC FOR HIERARCHICAL CLUSTERING

We detail the distance metrics used for hierarchical clustering below,

$$
\begin{aligned}
D(i,j) &= \lambda_1 D_{\text{center}}(i,j) + \lambda_2 D_{\text{area}}(i,j) + \lambda_3 D_{\text{shape}}(i,j) + \lambda_4 D_{\text{angle}}(i,j) + \lambda_5 D_{\text{merge}}(i,j), \\
D_{\text{center}}(i,j) &= \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}, \\
D_{\text{area}}(i,j) &= \left| l_i w_i - l_j w_j \right|, \\
D_{\text{shape}}(i,j) &= \left| (l_i / w_i) - (l_j / w_j) \right|, \\
D_{\text{angle}}(i,j) &= \left| (a_i + a_j)/2 - a_{\texttt{MBR}(i,j)} \right|, \\
D_{\text{merge}}(i,j) &= \left| l_i w_i + l_j w_j - l_{\texttt{MBR}(i,j)} w_{\texttt{MBR}(i,j)} \right|,
\end{aligned}
$$

where $D(i,j)$ represents the distance between rectangle i and j, $\texttt{MBR}(i,j)$ represents the minimum bounding rectangle (MBR) of rectangle i and j and λ represents the weight of each distance. Specifically, D_{center} measures the Euclidean distance between the center points of two rectangles; D_{area}, D_{shape} and D_{angle} separately measure the difference between the area, the aspect ratio, and the orientation of two rectangles; and D_{merge} measures the difference between the sum of the two rectangles area and their MBR area. The values of λ are determined empirically, detailed in Section 4.5.

A.3 MORE RECONSTRUCTION RESULTS

Figure 8 and 10 shows more of the 3D and 2D reconstruction results on the NYC dataset. Figure 9 and 11 shows more of the 3D and 2D reconstruction results on the Zurich dataset.

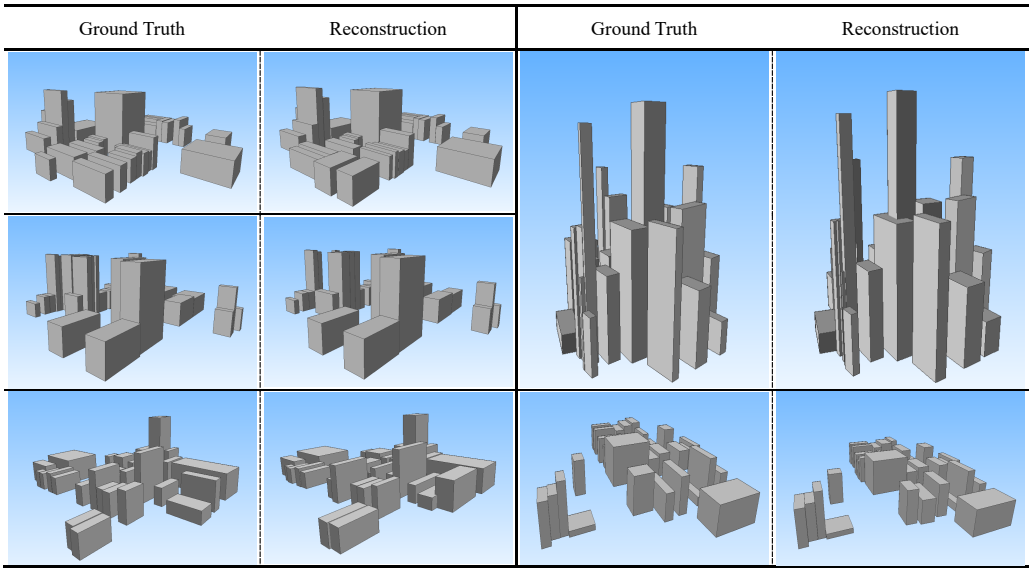

Figure 8: 3D reconstruction results of AETree trained on the NYC Dataset

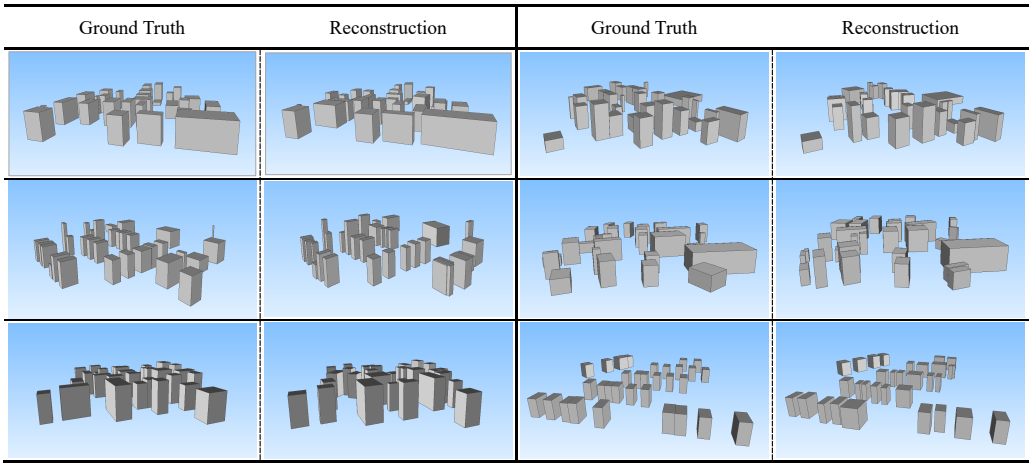

Figure 9: 3D reconstruction results of AETree trained on the Zurich Dataset

A.4 MORE GENERATION RESULTS

Figure 12 and 13 shows more of the unconditional 3D and 2D generation results on the NYC dataset. Figure 14 shows more of the unconditional 2D generation results on the Zurich dataset.

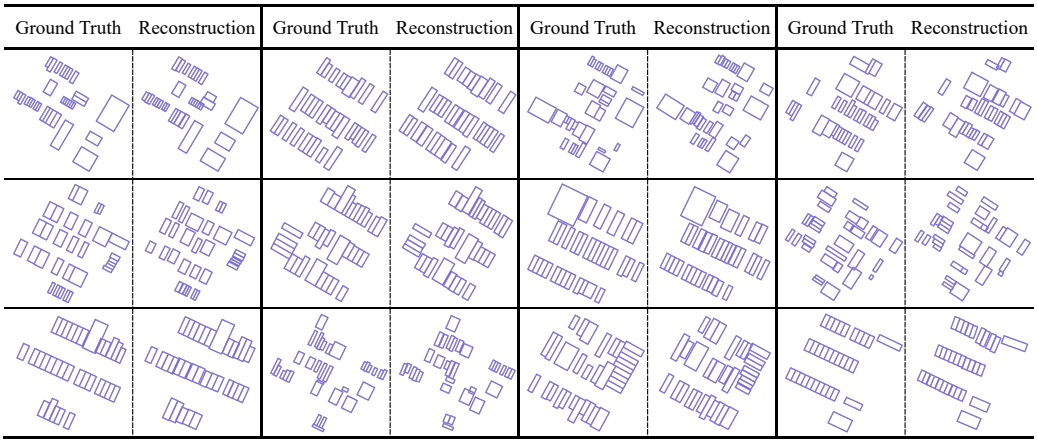

Figure 10: 2D reconstruction results of AETree trained on the NYC Dataset

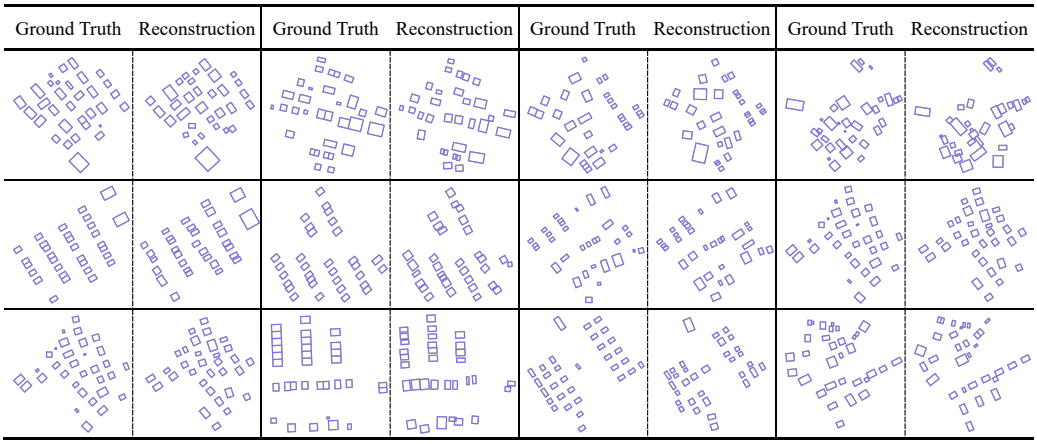

Figure 11: 2D reconstruction results of AETree trained on the Zurich Dataset

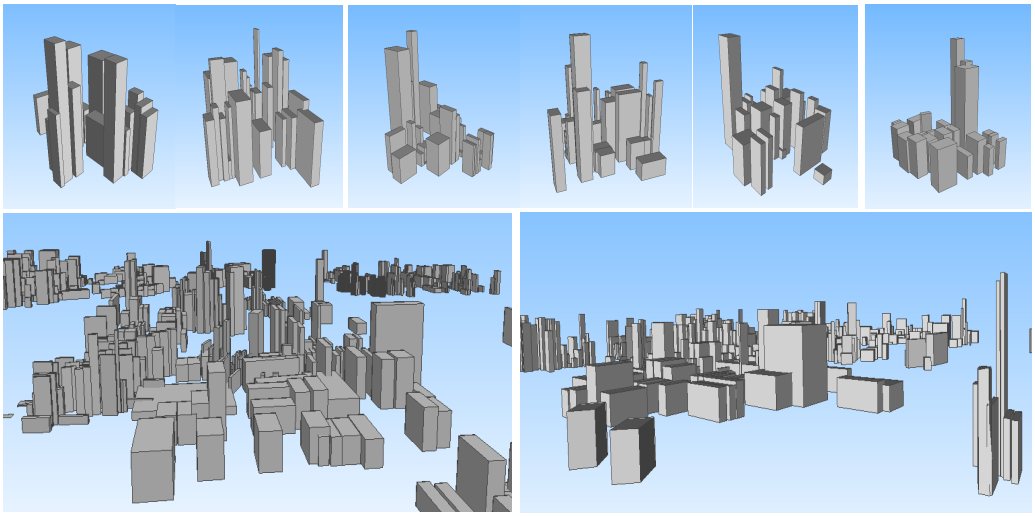

Figure 12: 3D generation results of AETree trained on the NYC Dataset

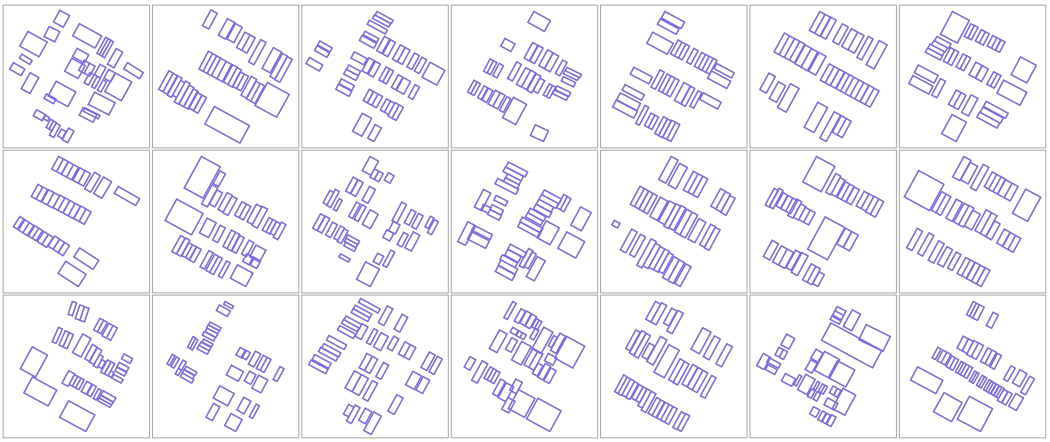

Figure 13: 2D generation results of AETree trained on the NYC Dataset

Figure 14: 2D generation results of AETree trained on the Zurich Dataset