Neural Turtle Graphics for Modeling City Road Layouts

Hang Chu[1,2,4] Daiqing Li[4] David Acuna[1,2,4] Amlan Kar[1,2,4] Maria Shugrina[1,2,4] Xinkai Wei[1,4]
Ming-Yu Liu[4] Antonio Torralba[3] Sanja Fidler[1,2,4]
[1]University of Toronto [2]Vector Institute [3]MIT [4]NVIDIA
{chuhang1122,davidj,amlan}@cs.toronto.edu, {daiqingl,mshugrina,xinkaiw,mingyul,sfidler}@nvidia.com, torralba@mit.edu

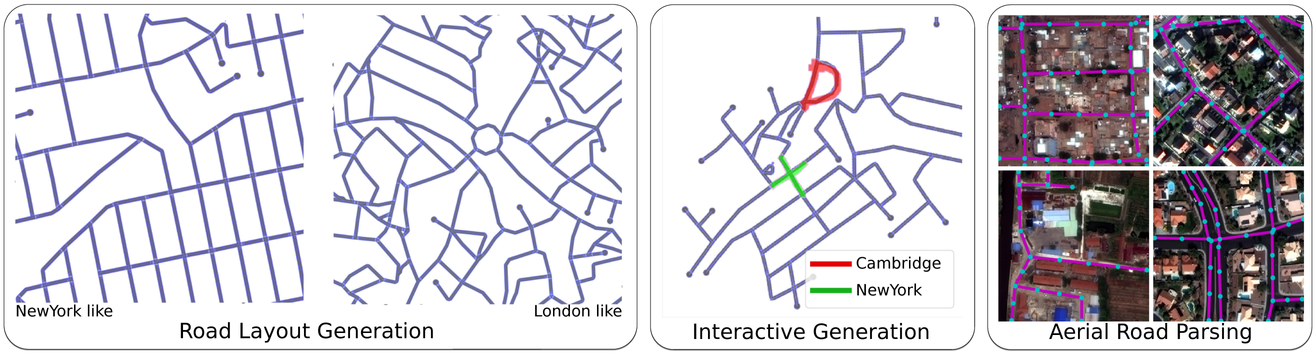


Figure 1: We introduce Neural Turtle Graphics (NTG), a deep generative model for planar graphs. In the Figure, we show NTG's applications to (interactive) city road layout generation and parsing.

Abstract

We propose Neural Turtle Graphics (NTG), a novel generative model for spatial graphs, and demonstrate its applications in modeling city road layouts. Specifically, we represent the road layout using a graph where nodes in the graph represent control points and edges in the graph represents road segments. NTG is a sequential generative model parameterized by a neural network. It iteratively generates a new node and an edge connecting to an existing node conditioned on the current graph. We train NTG on Open Street Map data and show that it outperforms existing approaches using a set of diverse performance metrics. Moreover, our method allows users to control styles of generated road layouts mimicking existing cities as well as to sketch parts of the city road layout to be synthesized. In addition to synthesis, the proposed NTG finds uses in an analytical task of aerial road parsing. Experimental results show that it achieves state-of-the-art performance on the SpaceNet dataset.

1. Introduction

City road layout modeling is an important problem with applications in various fields. In urban planning, extensive simulation of city layouts are required for ensuring that the final construction leads to effective traffic flow and connectivity. Further demand comes from the gaming industry where on-the-fly generation of new environments enhances user interest and engagement. Road layout generation also plays an important role for self-driving cars, where diverse virtual city blocks are created for testing autonomous agents.

Although the data-driven end-to-end learning paradigm has revolutionized various computer vision fields, the leading approaches [32] (e.g., the foundation piece in the commercially available CityEngine software) for city layout generation are still largely based on procedural modeling with hand-designed features. While these methods guarantee valid road topologies with user specified attribute inputs, the attributes are all hand-engineered and inflexible to use. For example, if one wishes to generate a synthetic city that resembles e.g. London, tedious manual tuning of the attributes is required in order to get plausible results. Moreover, these methods cannot trivially be used in aerial road parsing.

In this paper, we propose a novel generative model for city road layouts that learns from available map data. Our model, referred to as *Neural Turtle Graphics* (NTG) is inspired by the classical turtle graphics methodology [1] that progressively grows road graphs based on local statistics. We model the city road layout using a graph. A node in the graph represents a spatial control point of the road layout, while the edge represents a road segment. The proposed NTG is realized as an encoder-decoder architecture where the encoder is an RNN that encodes local incoming paths into a node and the decoder is another RNN that generates

Project page: https://nv-tlabs.github.io/NTG

[1] Turtle graphics is a technique for vector drawing, where a relative cursor (turtle) receives motion commands and leave traces on the canvas.

outgoing nodes and edges connecting an existing node to the newly generated nodes. Generation is done iteratively, by pushing newly predicted nodes onto a queue, and finished once all nodes are visited. Our NTG can generate road layouts by additionally conditioning on a set of attributes, thus giving control to the user in generating the content. It can also take a user specified partial sketch of the roads as input for generating a complete city road layout. Experiments with a comparison to strong baselines show that our method achieves better road layout generation performance in a diverse set of performance metrics. We further show that the proposed NTG can be used as an effective prior for aerial map parsing, particularly in cases when the imagery varies in appearance from that used in training. Fine-tuning the model jointly with CNN image feature extraction further improves results, outperforming all existing work on the Spacenet benchmark.

2. Related Work

Classical Work. A large body of literature exists on procedural modeling of streets. The seminal early work of [32] proposed an L-system which iteratively generates the map while adjusting parameters to conform to user guidance. This method became the foundation of the commercially available state-of-the-art CityEngine [1] software. Several approaches followed this line of work, exploiting user-created tensor fields [11], domain splitting [40], constraints stemming from the terrain [18, 8], and blending of retrieved examplars [6, 31, 16]. Methods that evolve a road network using constraints driven by crowd behaviour simulation have also been extensively studied [37, 33, 17].

Generative Models of Graphs. Graph generation with neural networks has only recently gained attention [41, 25, 35, 9]. [41] uses an RNN to generate a graph as a sequence of nodes sorted by breadth-first order, and predict edges to previous nodes as the new node is added. [35] uses a variational autoencoder to predict the adjacency and attribute matrices of small graphs. [25] trains recurrent neural network that passes messages between nodes of a graph, and generates new nodes and edges using the propagated node representation. Most of these approaches only predict graph topology, while in our work we address generation of spatial graphs. Producing valid geometry and topology makes our problem particularly challenging. Our encoder shares similarities with node2vec [19] which learns node embeddings by encoding local connectivities using random walks. Our work focuses on spatial graph generation and particularly on road layouts, thus different in scope and application.

Graph-based Aerial Parsing. Several work formulated road parsing as a graph prediction problem. The typical approach relies on CNN road segmentation followed by thinning [30]. To deal with errors in parsing, [30] proposes to reason about plausible topologies on an augmented graph as a shortest path problem. In [26], the authors treat local city patches as a simply connected maze which allows them to define the road as a closed polygon. Road detection then follows Polygon-RNN [10, 4] which uses an RNN to predict vertices of a polygon. [22] performs lane detection by predicting polylines in a top-down LIDAR view using a hierarchical RNN. Here, one RNN decides on adding new lanes, while the second RNN predicts the vertices along the lane. In our work, we predict the graph directly. Since our approach is local, it is able to grow large graphs which is typically harder to handle with a single RNN. Related to our work, [29, 12, 27, 4] annotate building footprints with a graph generating neural network. However, these works are only able to handle single cycle polygons.

Most related to our work is RoadTracer [7], which iteratively grows a graph based on image evidence and local geometry of the already predicted graph. At each step, RoadTracer predicts a neighboring node to the current active node. Local graph topology is encoded using a CNN that takes as input a rendering of the existing graph to avoid falling back. Our method differs in the encoder which in our case operates directly on the graph, and the decoder which outputs several outgoing nodes using an RNN which may better capture more complex road intersection topologies. Furthermore, while [7] relied on carefully designed dynamic label creation during training to mimic their test time graph prediction, our training regime is simple and robust to test time inference.

We also note that with some effort many of these work could be turned into generative models, however, ours is the first that showcases generative and interactive modeling of roads. Importantly, we show that NTG trained only on map data serves as an efficient prior for aerial road parsing. This cannot easily be done with existing work [7, 26] which all train a joint image and geometry representation.

3. Neural Turtle Graphics

We formulate the city road layout generation problem as a planar graph generation problem. We first introduce the notation in Sec. 3.1 and then describe our NTG model in Sec. 3.2. Aerial parsing, implementation details, training and inference are given in Sec. 3.3-Sec. 3.5, respectively.

3.1. Notation

Road Layout. We represent a city road layout using an undirected graph $G = \{V, E\}$, with nodes V and edges E. A node $\mathbf{v}_i \in V$ encodes its spatial location $[x_i, y_i]^T$, while an edge $e_{\mathbf{v}_i,\mathbf{v}_j} \in \{0, 1\}$ denotes whether a road segment connecting nodes $\mathbf{v}_i$ and $\mathbf{v}_j$ exists. City road graphs are planar since all intersections are present in V. We assume it is connected, *i.e.* there is a path in G between any two nodes in V. The coordinates x_i and y_i are measured in meters, relative to the city's world location.

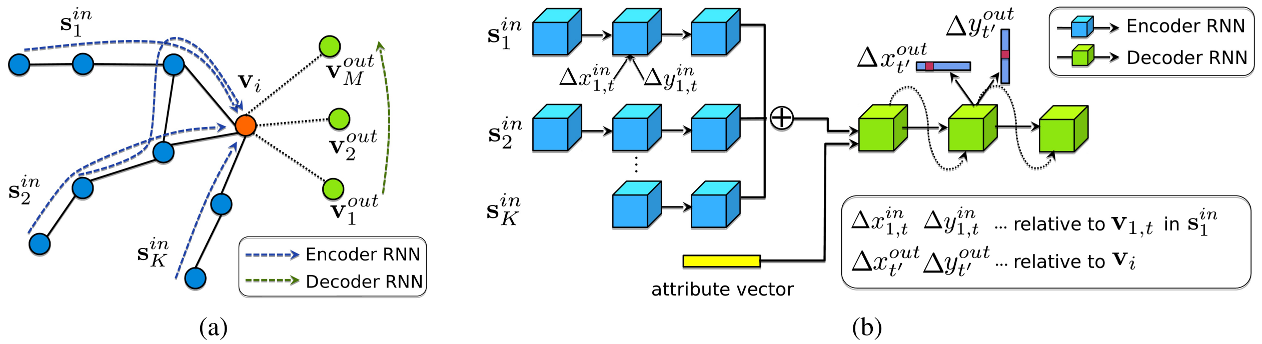


Figure 2: Illustration of the Neural Turtle Graphics (NTG) model. (a) depicts acyclic incoming paths $\{\mathbf{s}^{in}\}$ of an active node $\mathbf{v}_i$, each of which is encoded using an RNN encoder. NTG decoder then predicts a set of outgoing nodes $\{\mathbf{v}^{out}\}$. (b) shows the NTG's neural network architecture. First, the encoder GRU consumes the motion trajectory $\Delta\mathbf{x}^{in}$ of each incoming path. We produce an order-invariant representation by summing up the last-state hidden vectors across all paths. Next, the decoder produces "commands" to advance the turtle and produces new nodes. An optional attribute vector can be further added to the decoder depending on the task.

Incoming Paths. For a node $\mathbf{v}_i$, we define an *Acyclic Incoming Path* as an ordered sequence of unique, connected nodes which terminates at $\mathbf{v}_i$: $\mathbf{s}^{in}= \{\mathbf{v}_{i,1}, \mathbf{v}_{i,2}, ..., \mathbf{v}_{i,L}, \mathbf{v}_i\}$ where $e_{\mathbf{v}_{i,t},\mathbf{v}_{i,t+1}} = 1$ for each $1 \leq t < L$, and $e_{\mathbf{v}_{i,L},\mathbf{v}_i} = 1$, with L representing the length of the path. Since multiple different acyclic paths can terminate at $\mathbf{v}_i$, the set of these paths is denoted as $S_i^{in} := \{\mathbf{s}_k^{in}\}$.

Outgoing Nodes. We define $V_i^{out} := \{\mathbf{v}_j : \mathbf{v}_j \in V \wedge e_{\mathbf{v}_i,\mathbf{v}_j} = 1\}$, *i.e.* as the set of nodes with an edge to $\mathbf{v}_i$.

3.2. Graph Generation

We learn to generate graphs in an iterative manner. The graph is initialized with a root node and a few nodes connected to it, which are used to initialize a queue Q of unvisited nodes. In every iteration, an unvisited node from Q is picked to be expanded (called *active node*). Based on its *current local topology*, an encoder model generates a latent representation, which is used to generate a set of neighboring nodes using a decoder model. These generated nodes are pushed to Q. The node to be expanded next is picked by popping from Q, until it is empty.

By construction, an active node $\mathbf{v}_i$ has at least one neighbor node in the graph. NTG extracts a representation of its local topology by encoding *incoming paths* S_i^{in} (of maximum length L) and uses the representation to generate a set of *outgoing nodes* V_i^{out} (if any) with edges to $\mathbf{v}_i$. These paths are encoded in an order-invariant manner and the resulting latent representation is used to generate a set of outgoing nodes V_i^{out}. NTG performs the encoding and decoding to generate the graph as described above with an encoder-decoder neural network. Fig. 2(a) visualizes the process, while Fig. 2(b) illustrates the encoder-decoder neural architecture, which is described in detail in the following sections.

NTG Encoder. We encode a single incoming path $\mathbf{s}^{in}$ into node $\mathbf{v}_i$ with a zero-initialized, bidirectional GRU [13]. The input to the GRU while processing the t^{th} node in the path is the motion vector between the nodes $\mathbf{v}_{i,t}^{in} \in \mathbf{s}^{in}$ and $\mathbf{v}_{i,t+1}^{in} \in \mathbf{s}^{in}$ in the path; *i.e.*, $[\Delta x_{i,t}^{in}, \Delta y_{i,t}^{in}]^T = [x_{i,t+1}^{in}, y_{i,t+1}^{in}]^T - [x_{i,t}^{in}, y_{i,t}^{in}]^T$. This offset could be encoded as a discrete or continuous value, as discussed in Sec. 3.4. The final latent representation $\mathbf{h}_{\text{enc}}$ for all paths is computed by summing the last hidden states of each path. Optionally, we append an attribute vector $\mathbf{h}_{\text{attr}}$ to the latent representation. For example, the attribute could be an embedding of a one-hot vector, encoding the city identity. This enables NTG to learn an embedding of city, enabling conditional generation. The final encoding is their concatenation $[\mathbf{h}_{\text{enc}}, \mathbf{h}_{\text{attr}}]$.

Sampling Incoming Paths. During training, for an active node $\mathbf{v}_i$ we use a subset of S_i^{in} by sampling K random walks (without repetition) starting from $\mathbf{v}_i$, such that each random walk visits at most L different nodes. We find this random sampling to lead to a more robust model as it learns to generate from incomplete and diverse input representations. Optionally, we can also feed disconnected adjacent nodes as additional input. We found this to perform similarly in the task of road modeling due to high connectivity in data.

Decoder. We decode the outgoing nodes V_i^{out} with a decoder GRU. The recurrent structure of the decoder enables capturing local dependencies between roads such as orthogonality at road intersections. We independently predict $\Delta x_{t'}^{out}$ and $\Delta y_{t'}^{out}$ for an outgoing node $\mathbf{v}_{t'}^{out}$, indicating a new node's relative location w.r.t. $\mathbf{v}_i$. Additionally, we predict a binary variable which indicates whether another node should be generated. At generation time we check overlap between the new node and existing graph with a $5m$ threshold to produce loops. Optionally, we predict the edge type between $(\mathbf{v}_i, \mathbf{v}_{t'}^{out})$, *i.e.* minor or major road, using a categorical variable. The hidden state $\mathbf{h}_{t'}$ of the decoder is updated as:

$$\mathbf{h}_{t'+1} = \text{GRU}(\mathbf{h}_{t'} \mid \mathbf{h}_{\text{enc}}, \mathbf{h}_{\text{attr}}, \Delta\mathbf{x}_{t'}^{out}) \qquad (1)$$

3.3. Aerial Road Parsing

Parsing with Map Prior. The dominant approaches to parse roads from aerial imagery have trained CNNs to produce a probability (likelihood) map, followed by thresholding and thinning. This approach typically results in maps with holes or false positive road segments, and heuristics are applied to postprocess the topology. We view a NTG model trained to generate city graphs (*i.e. not trained* for road parsing) as a prior, and use it to postprocess the likelihood coming from the CNN. Starting from the most confident intersection node as the root node, we push all its neighbors into the queue of unvisited nodes, and then use NTG to expand the graph. At each decoding step, we multiply the likelihood from CNN with the prior produced by NTG, and sample output nodes from this joint distribution. The end of sequence is simply determined by checking whether the maximum probability of a new node falls below a threshold (0.05 in our paper).

Image-based NTG. We also explore *explicitly training* NTG for aerial parsing. We condition on image features by including predictions from a CNN trained to parse aerial images in the attribute vector $\mathbf{h}_{attr}$. In practice, we initialize the graph obtained by thresholding and thinning the ouputs of the CNN, and use the trained image-based NTG on top.

3.4. Implementation Details

We exploit the same NTG model in both tasks of city generation and road detection. Depending on the task, we empirically find that the best parameterization strategy varies. For city generation, we use discrete Δx, Δy with resolution of $1m$ for both encoder and decoder, where x points to east and y points to north. Here, Δx and Δy are limited to $[-100 : 100]$, indicating that the largest offset in either direction is $100m$. The discrete Δx and Δy values are given associated embeddings (resembling words in language models), which are concatenated to generate the input to the encoder at every step. For road detection, we use continuous polar coordinates in the encoder, where the axis is rotated to align with the edge from the previous to the current node. This forms rotation invariant motion trajectories that help detecting roads with arbitrary orientation. The decoder always uses a discrete representation. We encode and decode the coordinates x and y independently. We find that this yields similar results compared to joint prediction, while significantly saving training memory and model capacity. 500 hidden units are used in encoder and decoder GRUs.

3.5. Learning & Inference

Inference. At each inference step, we pop a node from the queue Q, encode its existing incoming paths, and generate a set of new nodes. For each new node, we check if it is in the close proximity of an existing node in the graph. If the distance to an existing node is below a threshold ϵ ($5m$ in our paper), we do not add the new node to the queue. Instead, an edge is included to connect the current node to the existing node. This enables the generation of cycles in the graph. We also find the maximum node degree, maximum node density, and minimum angle between two edges in the training set, and ensure our generation does not exceed these limits. We refer to supplemental material for their effects.

	Country	City	Node	Edge	Area	Length
RoadNet	13	17	233.6k	262.1k	170.0km^2	7410.7km
SpaceNet	4	4	115.8k	106.9k	122.3km^2	2058.4km

Table 1: Dataset statistics of RoadNet and SpaceNet [2].

Learning. At training time, K incoming paths for each $\mathbf{v}_i$ are sampled, and we learn to predict *all* of its neighboring nodes. We enforce an order in decoding the nodes, where we sort nodes counter-clockwise to form a sequence. The ordering saves having to solve an assignment problem to compute the loss function. Our model is trained using ground truth map data with teacher-forcing [39], using a cross entropy loss for each of the output nodes. The networks are optimized using Adam [24] with a learning rate of 1e-3, weight decay of 1e-4, and gradient clipping of 1.0.

4. Experiments

We demonstrate NTG on three tasks: city road layout generation, satellite road parsing, and environment simulation.

4.1. City Road Layout Generation

RoadNet Dataset. We collected a real-world road dataset from OpenStreetMap (OSM) to facilitate this task. In particular, we selected 17 unique cities across continents and gathered all road markers. OSM, being crowd-sourced, often has incomplete markers in underpopulated areas. To alleviate this, we manually select the most densely annotated $10km^2$ region within each city. These constitute our final RoadNet dataset. Table 1 shows the statistics.

4.1.1 Metrics

The goals of road layout generation are to create road networks that are: **a)** Perceptually plausible and preserve a meaningful city style, and **b)** Diverse. We use three broad categories of automatic metrics to evaluate city generation:

Perceptual: For every node, we render the graph in a $300m$ neighborhood centered around it on a canvas. With their perceptual features extracted from an InceptionV3 network [36], we compute the Fréchet Inception Distance (FID) [21] between the synthesized roads and ground truth maps for each city. To ensure a meaningful FID, we adapt InceptionV3, which has originally been trained on natural images, to road drawings, by finetuning it to predict city ids on our dataset. This yields a 90.27% accuracy, indicating effective capture of style across cities in the network.

Urban Planning [5]: We measure four common urban planning features reflecting city style: **1.** Node *density* within a neighborhood of $100m$, $200m$, and $300m$. **2.** *Connectivity* as reflected by the degrees of nodes. **3.** *Reach* as the total length of accessible roads within a distance of $100m$, $200m$, and $300m$. **4.)** Transportation *convenience* as the ratio of

Metric / Method	*Perceptual*					*Urban Planning*					*Diversity*
	mp1 10^{-1}	mp2 10^{0}	pa 10^{-1}	fc 10^{1}	rate	densi. 10^{1}	conne. 10^{-2}	reach 10^{5}	conve. 10^{-3}	rate	
GraphRNN-2D [41, 7]	7.12	6.35	8.45	16.15	25.0	51.58	4.61	45.11	6.72	43.7	44.26
PGGAN [23]	1.98	2.15	5.34	10.51	63.2	45.77	19.48	4.33	2.94	58.9	5.95
CityEngine-5k [1]	2.74	2.71	8.34	14.78	47.1	13.59	21.66	7.61	16.66	51.7	45.86
CityEngine-10k [1]	2.55	2.56	8.23	14.17	48.9	12.43	21.79	7.05	16.82	52.1	46.00
NTG-vanilla	2.63	2.33	4.05	9.17	66.0	8.69	**1.87**	8.99	3.06	86.5	41.27
NTG-enhance	**1.52**	**1.34**	**2.83**	**6.76**	**77.3**	**3.76**	1.97	**4.13**	**1.86**	**92.4**	42.09

Table 2: Perceptual domain-adapted FIDs ({maxpool1,maxpool2,pre-aux,fc}, lower is better), Urban Planning feature differences ({density,connectivity,reach,convenience}, lower is better), and Diversity evaluation of city generation. Ratings (higher is better) are computed by averaging with scales {10,10,10,20} for perceptual and {60,30,50,20} for urban planning. Extremely low Diversity indicates incapability of creating new cities.

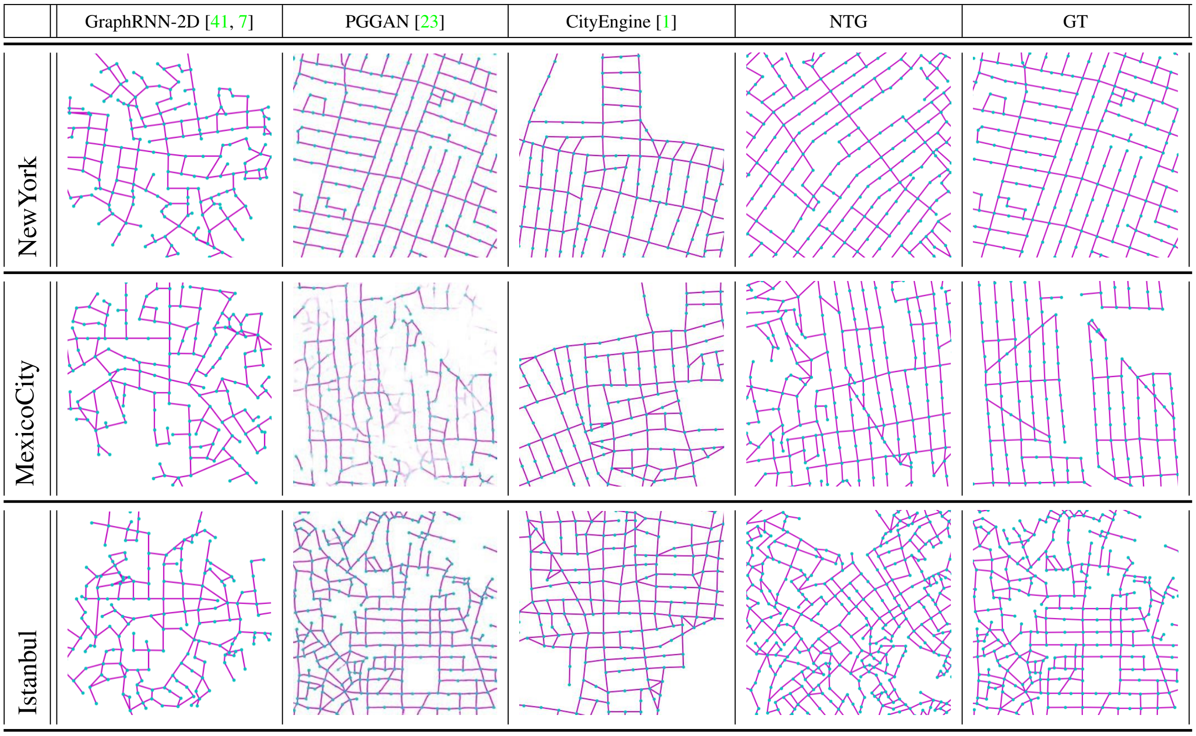


Figure 3: Qualitative examples of city road layout generation. GraphRNN-2D generates unnatural structures and fails to capture city style. PGGAN is unable to create new cities by either severely overfitting, or producing artifacts. CityEngine produces less style richness due to its fixed rule-based synthesis algorithm. NTG is able to both capture the city style and creating new cities.

the euclidean distance over the Dijkstra shortest path for node pairs that are more than 500*m* away. We also compute the Fréchet distance between normal distributions computed from the concatenation of the Urban Planning features of real and generated maps.

Diversity metric: We measure the ability to create novel cities by computing the overlap between a real and generated city as the percentage of road in one graph falling outside the 10*m* vicinity of the road in the other graph, and vice versa. We compare this Chamfer-like distance against all ground truth maps and report the average lowest value.

4.1.2 Results

We compare the following methods:

- **GraphRNN-2D [41, 7]**: We enhance the GraphRNN model by introducing extra branches to encode/decode node coordinates and city id. We add a CNN that takes into account local rendering of existing graph as in [7] and add checks to avoid invalid edge crossing during inference.
- **PGGAN [23]**: We train to generate images of road layouts at a resolution of 256×256. We use our trained InceptionV3 network to classify samples into cities to compute city-wise metrics. For computing the graph-related metrics we convert images to graphs by thresholding and thinning.
- **CityEngine [1]**: CityEngine is a state-of-the-art software for synthesizing cities based on an optimized, complex rule-based L-system [32]. By default, it only offers limited templates and is incapable of generating new cities. To enhance CityEngine, we use its provided control interface and ex-

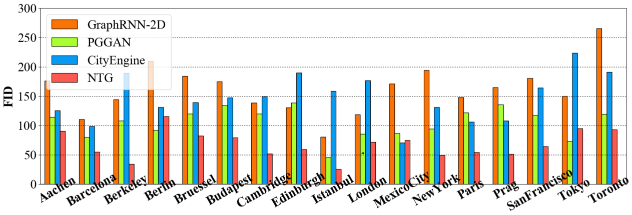


Figure 4: City-wise FID (fc) of different methods.

haustively search over its attribute space by enumerating combinations of important control parameters such as angles, bending specifications, and crossing ratios. We then predict city probabilities using the InceptionV3 network, and select the highest ranking $10km^2$ as the result for each city.

- **NTG**: NTG begins with a root node with its edges. We evaluate NTG with a random root (NTG-vanilla), as well as with a pre-stored high connectivity root (NTG-enhance).

Tab. 2 and Fig. 3 show quantitative and qualitative results, respectively. Quantitatively, NTG outperforms baselines across all metrics. GraphRNN-2D fails to capture various city styles and frequently produces unnatural structures. This is due to its sequential generative nature that depends on Breadth First Search. The RNN that encodes the full history fails to capture coherent structures since consecutive nodes may not be spatially close due to BFS. PGGAN [23] produces sharp images with occasional artifacts that are difficult to convert into meaningful graphs. Samples from PGGAN are severely overfit as reflected by the Diversity metric, indicating its inability to create new cities. Moreover, PGGAN also suffers from mode-collapse and memorizes a portion of data. This imbalance of style distribution leads to worse perceptual FIDs. With our enhancement (exhaustive search), CityEngine [1] is able to capture certain cities' style elements: especially the node density. However, it has less topological variance and style richness due to its fixed set of rules. Expanding its search from $5000km^2$ (CityEngine-5k) to $10000km^2$ (CityEngine-10k) of generated maps does not lead to significant improvements, while requiring double the amount of computation. NTG is able to create new cities, while better capturing style in most cities as shown in Fig. 4.

Fig. 5 digs into NTG's generated maps, showing that NTG learns to remember local road patterns. As the graph grows, different local topologies are organically intertwined to produce new cities. Fig. 6 shows the effect of two important hyper-parameters: maximum number of paths K and maximum incoming path length L. Results show that reconstruction quality is determined by L, while K and L both affect inference time. Training with longer and more paths does not necessarily improve perceptual quality, since generation starts from a single root node without long paths.

We further demonstrate our approach by having NTG predict two types of roads, *i.e.* major and minor roads. Results are shown in Fig. 8, showing that NTG easily generalizes to a more complex modeling problem.

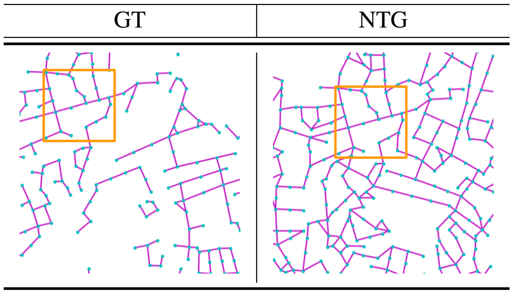
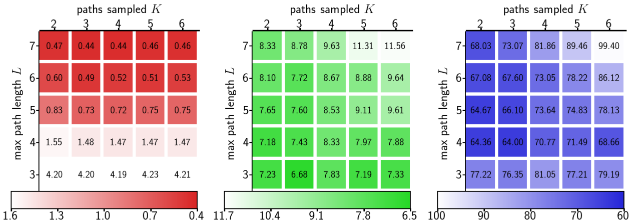


Figure 5: NTG creates new city road layouts in a combinatorial manner. Local patterns as shown by orange boxes are remembered, and then intertwined to create novel structures.



max path length L \ paths sampled K	2	3	4	5	6
7	0.47	0.44	0.44	0.46	0.46
6	0.60	0.49	0.52	0.51	0.53
5	0.83	0.73	0.72	0.75	0.75
4	1.55	1.48	1.47	1.47	1.47
3	4.20	4.20	4.19	4.23	4.21

max path length L \ paths sampled K	2	3	4	5	6
7	8.33	8.78	9.63	11.31	11.56
6	8.10	7.72	8.67	8.88	9.64
5	7.65	7.60	8.53	9.11	9.61
4	7.18	7.43	8.33	7.97	7.88
3	7.23	6.68	7.83	7.19	7.33

max path length L \ paths sampled K	2	3	4	5	6
7	68.03	73.07	81.86	89.46	99.40
6	67.08	67.60	73.05	78.22	86.12
5	64.67	66.10	73.64	74.83	78.13
4	64.36	64.00	70.77	71.49	68.66
3	77.22	76.35	81.05	77.21	79.19

Figure 6: Effect of sampled paths K and maximum path length L on reconstruction quality in meters (red), inference time in seconds per km^2 (green), and FID-fc (blue).

Interactive Synthesis. We showcase an application for interactive road layout generation where a user chooses from a palette of cities and provides local topology priors by sketching. We match the user's input with pre-stored node templates to form the root node. To allow generating multiple components on the same canvas, we simply modify the NTG inference procedure to iterate through multiple queues in parallel. Fig. 7 shows examples of the generation process.

Beyond Road Layouts. In Appendix, we show results on using NTG's multipath paradigm for learning effective representation of complex shapes, such as multi-stroke hand drawings. This shows potential as a general purpose spatial graph generative model beyond the city road modeling.

4.2. Satellite Road Parsing

SpaceNet Dataset. While several datasets have been presented for road detection [2, 14, 38, 7], we use SpaceNet [2] for its large scale, image quality, and open license. To facilitate consistent future benchmarking, we reformat the raw data into an easy-to-use version with consistent tile size in metric space. Tab. 1 shows its statistics. We split tiles of each city into train-validation-test with a 4-1-1 ratio.

4.2.1 Metrics

Average Path Length Similarity (APLS) has been shown to be the best metric to reflect routing properties [2]. Between two graphs, APLS is defined as

$$\text{APLS} = 1 - \frac{1}{N_p} \sum_{p_{\mathbf{v}_1\mathbf{v}_2}<\infty} min \left\{ 1, \frac{|p_{\mathbf{v}_1\mathbf{v}_2} - p_{\mathbf{v}_1'\mathbf{v}_2'}|}{p_{\mathbf{v}_1\mathbf{v}_2}} \right\}$$

where $\mathbf{v}$ denotes a source graph node, $\mathbf{v}'$ as its closest on-road point in the target graph if such a point exists within a

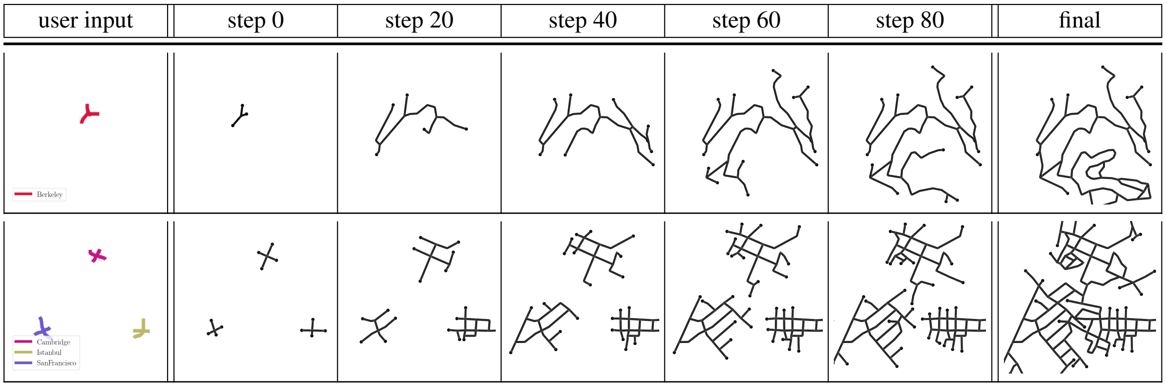


Figure 7: Examples of interactive city road layout generation via user sketching and local style selection.

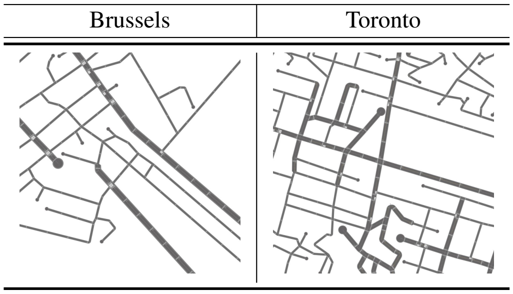


Figure 8: NTG can be easily extended to generate road type.

	IOU	F1	APLS
DeepRoadMapper [30]	45.02	62.08	51.49
RoadExtractor [7]	52.91	69.20	57.38
RoadTracer [7]	10.23	18.56	48.55
RoadTracer-30 [7]	48.29	65.13	42.94
FCN [20, 28]	51.09	67.63	56.56
DLA+STEAL [42, 3]	58.96	74.18	71.04
NTG-P ([30]'s CNN)	50.58	67.18	55.87
NTG-P ([7]'s CNN)	51.62	68.09	58.79
NTG-P (DLA+STEAL)	59.29	74.44	70.99
NTG-I (DLA+STEAL)	**63.15**	**77.42**	**74.97**

Table 3: Comparison of methods on the standard SpaceNet split.

	IOU	F1	APLS
RoadExtractor [7]	20.51	34.03	43.06
DLA+STEAL [42, 3]	33.94	50.68	56.15
NTG-P (DLA+STEAL)	**35.16**	**52.02**	**57.89**

Table 4: SpaceNet evaluation on unseen city by holding one city out in training. Without finetuning, the RoadNet pretrained NTG-P is able to improve over DLA+STEAL.

buffer range ($5m$), N_p number of paths. Here, $p_{\mathbf{v}_1\mathbf{v}_2}$ denotes the Dijkstra shortest path length between two nodes, and has infinite value if no path exists. We also exchange source and target graphs to establish metric symmetry. To ensure even node distribution, graphs are RDP-simplified [34, 15] and uniformly subdivided with $30m$ maximum edge length. While we use APLS as our main metric, we also report conventional pixel-wise IOU and F1 score as references, even though they are less desirable as revealed in [2].

4.2.2 Results

We compare three categories of methods:

- **Prior art**: We evaluate DeepRoadMapper [30], RoadExtractor [7], and RoadTracer [7]. RoadTracer requires additional starting points as input. We use the most likely pixel predicted by their CNN, as well as 30 points randomly selected from ground truth (RoadTracer-30).
- **Stronger CNNs**: We explore more powerful CNN architectures. We train an FCN with a ResNet backbone [20, 28], as well as a CNN using DLA [42] with STEAL [3]. To obtain the graph we use standard thinning and geodesic sorting.
- **NTG**: We evaluate both the parsing NTG (NTG-P) that is only trained on RoadNet and acts as a topological prior and image-based NTG (NTG-I) that is trained on SpaceNet.

Table 3 and Figure 9 present SpaceNet results. It can be seen that our method outperforms baselines in all metrics. The DLA+STEAL CNN produces cleaner predictions that focus on road. NTG-P trained only with RoadNet is able to successfully parse graph structure. Using NTG-I that further takes CNN output as input achieves the best result. We also experiment the RoadNet trained NTG-P with CNNs from prior art [30, 7]. It can be seen that the city topology knowledge of NTG makes it a better graph extractor compared to hand-crafted postprocessings in [30, 7], especially in terms of APLS. For NTG-P with DLA+STEAL we notice it has similar performance as standard graph extraction. This is because DLA+STEAL prediction has high confidence as it is trained and tested with same cities that have similar visual appearance. We therefore further experiment with one city held-out to simulate road parsing in unseen cities. Results are presented in Table 4. It can be seen that NTG-P is able to further improve the result, demonstrating the effectiveness of generative road layout knowledge learnt from RoadNet. We conduct 4-fold evaluation holding out each city per fold, and report the average result.

4.3. Environment Simulation

We further showcase a novel application that combines our two tasks in Figure 10. We propose to directly convert a satellite image into simulation-ready environments, which may be important for testing autonomous vehicles in the future. First, we detect roads in the satellite image with

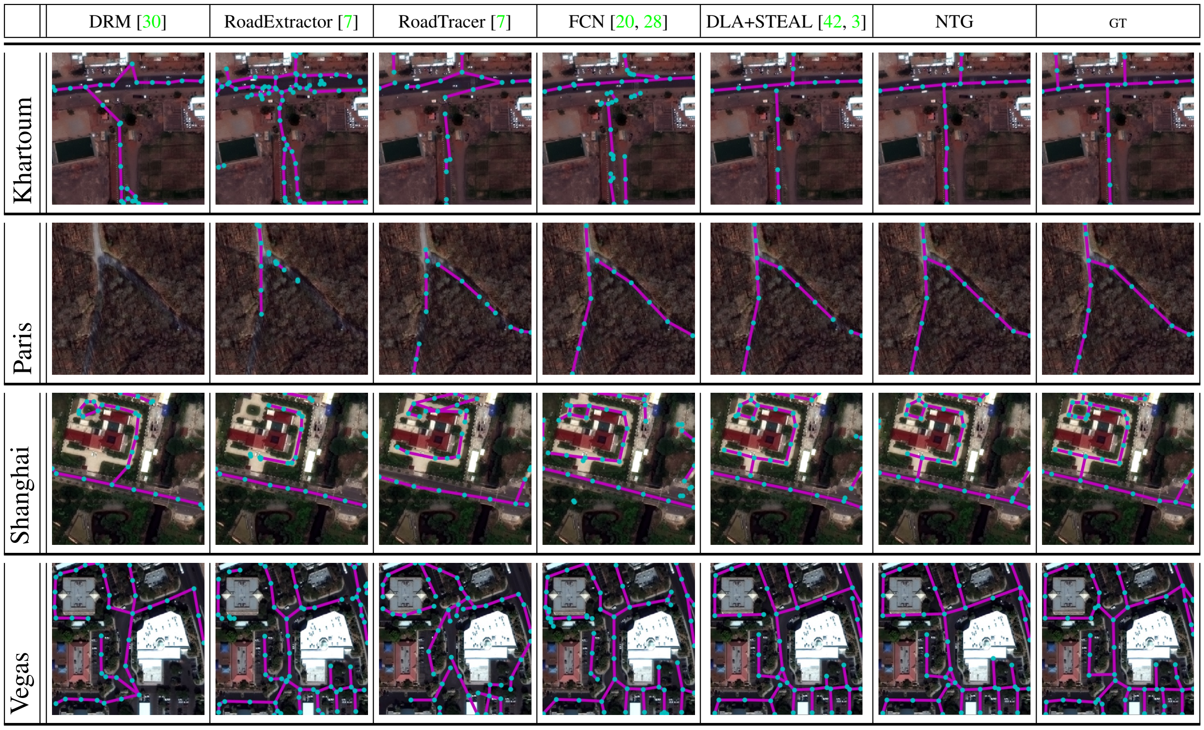


Figure 9: Qualitative examples of SpaceNet road parsing.

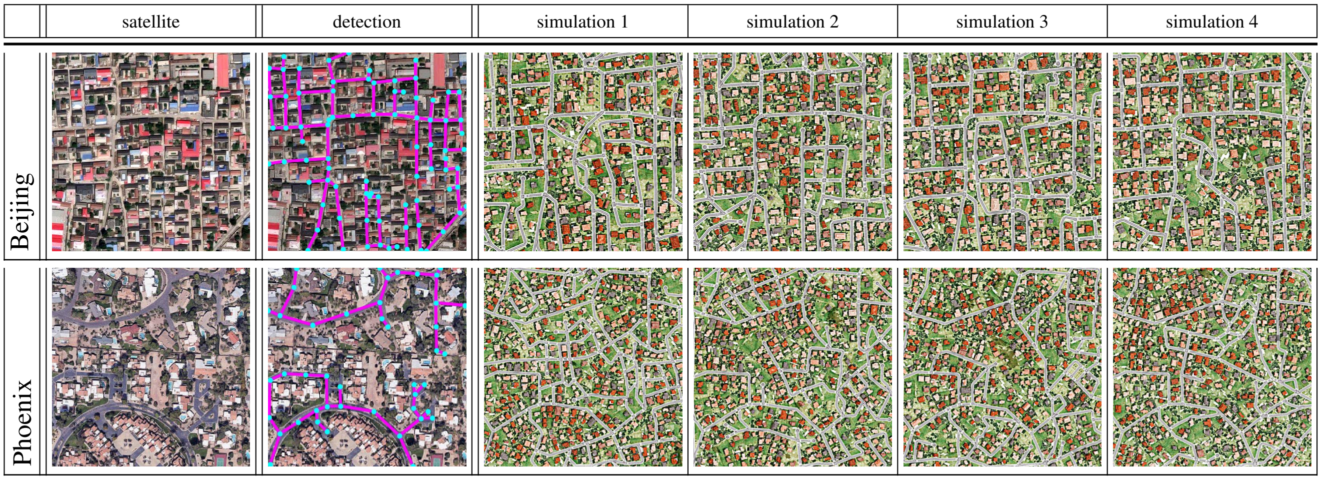


Figure 10: Sat2Sim: converting satellite image into a series of simulated environments. Buildings and vegetation added via [1].

NTG, giving us an initial graph. Then, we exploit our generative model to propose plausible variations. This is done by pushing all single-connection nodes in the parsed graph into our generative queue, and running the NTG generative process to expand the graph. We directly make use of the NTG model trained for city generation and choose a random city id for each run. This has two main advantages. First, it is fully automatic and only requires a low-cost satellite image as input. Second, it provides a set of plausible variations of the environment (city) instead of a static one, which could eventually enable training more robust agents. For visualization, we additionally add buildings and tree via [1], showing plausible and diverse simulation-ready cities.

5. Conclusion

In this paper, we proposed Neural Turtle Graphics for generating large spatial graphs. NTG takes the form of an encoder-decoder neural network which operates on graphs locally. We showcased NTG on generating plausible new versions of cities, interactive generation of city road layouts, as well as aerial road parsing. Furthermore, we combined the two tasks of aerial parsing and generation, and highlighted NTG to automatically simulate new cities for which it has not seen any part of the map during training time. In future work, we aim to tackle generation of other city elements such as buildings and vegetation.

References

[1] Esri: Cityengine. https://www.esri.com/en-us/arcgis/products/esri-cityengine. 2, 5, 6, 8

[2] Spacenet dataset. https://spacenetchallenge.github.io/. 4, 6, 7

[3] David Acuna, Amlan Kar, and Sanja Fidler. Devil is in the edges: Learning semantic boundaries from noisy annotations. In *CVPR*, 2019. 7, 8

[4] David Acuna, Huan Ling, Amlan Kar, and Sanja Fidler. Efficient interactive annotation of segmentation datasets with polygon-rnn++. In *CVPR*, 2018. 2

[5] Sawsan AlHalawani, Yong-Liang Yang, Peter Wonka, and Niloy J Mitra. What makes london work like london? In *Computer Graphics Forum*, 2014. 4

[6] Daniel G Aliaga, Carlos A Vanegas, and Bedrich Benes. Interactive example-based urban layout synthesis. In *SIGGRAPH Asia*, 2008. 2

[7] Favyen Bastani, Songtao He, Sofiane Abbar, Mohammad Alizadeh, Hari Balakrishnan, Sanjay Chawla, Sam Madden, and David DeWitt. Roadtracer: Automatic extraction of road networks from aerial images. In *CVPR*, 2018. 2, 5, 6, 7, 8

[8] Jan Benes, Alexander Wilkie, and Jaroslav Křivánek. Procedural modelling of urban road networks. *Computer Graphics Forum*, 2014. 2

[9] Aleksandar Bojchevski, Oleksandr Shchur, Daniel Zügner, and Stephan Günnemann. Netgan: Generating graphs via random walks. In *ICML*, 2018. 2

[10] Lluis Castrejon, Kaustav Kundu, Raquel Urtasun, and Sanja Fidler. Annotating object instances with a polygon-rnn. In *CVPR*, 2017. 2

[11] Guoning Chen, Gregory Esch, Peter Wonka, Pascal Müller, and Eugene Zhang. Interactive procedural street modeling. *TOG*, 2008. 2

[12] Dominic Cheng, Renjie Liao, Sanja Fidler, and Raquel Urtasun. Darnet: Deep active ray network for building segmentation. In *CVPR*, 2019. 2

[13] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv:1412.3555*, 2014. 3

[14] Ilke Demir, Krzysztof Koperski, David Lindenbaum, Guan Pang, Jing Huang, Saikat Basu, Forest Hughes, Devis Tuia, and Ramesh Raska. Deepglobe 2018: A challenge to parse the earth through satellite images. In *CVPR Workshops*, 2018. 6

[15] David H Douglas and Thomas K Peucker. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *Cartographica: the international journal for geographic information and geovisualization*, 1973. 7

[16] Arnaud Emilien, Ulysse Vimont, Marie-Paule Cani, Pierre Poulin, and Bedrich Benes. Worldbrush - interactive example-based synthesis of procedural virtual worlds. *TOG*, 2015. 2

[17] Tian Feng, Lap-Fai Yu, Sai-Kit Yeung, Kangkang Yin, and Kun Zhou. Crowd-driven mid-scale layout design. *TOG*, 2016. 2

[18] Eric Galin, Adrien Peytavie, Éric Guérin, and Bedrich Benes. Authoring hierarchical road networks. *Computer Graphics Forum*, 2011. 2

[19] Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In *KDD*, 2016. 2

[20] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016. 7, 8

[21] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium. In *NIPS*, 2017. 4

[22] Namdar Homayounfar, Wei-Chiu Ma, Shrinidhi Kowshika Lakshmikanth, and Raquel Urtasun. Hierarchical recurrent attention networks for structured online maps. In *CVPR*, 2018. 2

[23] Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen. Progressive growing of gans for improved quality, stability, and variation. In *ICLR*, 2018. 5, 6

[24] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv:1412.6980*, 2014. 4

[25] Yujia Li, Oriol Vinyals, Chris Dyer, Razvan Pascanu, and Peter Battaglia. Learning deep generative models of graphs. *arXiv:1803.03324*, 2018. 2

[26] Zuoyue Li, Jan Dirk Wegner, and Aurélien Lucchi. Polymapper: Extracting city maps using polygons. *arXiv:1812.01497*, 2018. 2

[27] Huan Ling, Jun Gao, Amlan Kar, Wenzheng Chen, and Sanja Fidler. Fast interactive object annotation with curve-gcn. In *CVPR*, 2019. 2

[28] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *CVPR*, 2015. 7, 8

[29] Diego Marcos, Devis Tuia, Benjamin Kellenberger, Lisa Zhang, Min Bai, Renjie Liao, and Raquel Urtasun. Learning deep structured active contours end-to-end. In *CVPR*, pages 8877–8885, 2018. 2

[30] Gellért Máttyus, Wenjie Luo, and Raquel Urtasun. Deeproadmapper: Extracting road topology from aerial images. In *ICCV*, 2017. 2, 7, 8

[31] G Nishida, I Garcia-Dorado, and D G Aliaga. Example-driven procedural urban roads. *Computer Graphics Forum*, 2015. 2

[32] Yoav I H Parish and Pascal Müller. Procedural modeling of cities. In *SIGGRAPH*, 2001. 1, 2, 5

[33] Chi-Han Peng, Yong-Liang Yang, Fan Bao, Daniel Fink, Dong-Ming Yan, Peter Wonka, and Niloy J Mitra. Computational network design from functional specifications. *TOG*, 2016. 2

[34] Urs Ramer. An iterative procedure for the polygonal approximation of plane curves. *Computer Graphics and Image Processing*, 1972. 7

[35] Martin Simonovsky and Nikos Komodakis. Graphvae: Towards generation of small graphs using variational autoencoders. In *ICANN*, 2018. 2

[36] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *CVPR*, 2016. 4

[37] Carlos A Vanegas, Daniel G Aliaga, Bedrich Benes, and Paul Waddell. Interactive design of urban spaces using geometrical and behavioral modeling. *TOG*, 2009. 2

[38] Shenlong Wang, Min Bai, Gellert Mattyus, Hang Chu, Wenjie Luo, Bin Yang, Justin Liang, Joel Cheverie, Sanja Fidler, and Raquel Urtasun. Torontocity: Seeing the world with a million eyes. In *ICCV*, 2017. 6

[39] Ronald J Williams and David Zipser. A learning algorithm for continually running fully recurrent neural networks. *Neural computation*, 1989. 4

[40] Yong-Liang Yang, Jun Wang, Etienne Vouga, and Peter Wonka. Urban pattern - layout design by hierarchical domain splitting. *TOG*, 2013. 2

[41] Jiaxuan You, Rex Ying, Xiang Ren, William Hamilton, and Jure Leskovec. Graphrnn: Generating realistic graphs with deep auto-regressive models. In *ICML*, 2018. 2, 5

[42] Fisher Yu, Dequan Wang, Evan Shelhamer, and Trevor Darrell. Deep layer aggregation. In *CVPR*, 2018. 7, 8

Neural Turtle Graphics for Modeling City Road Layouts
Appendix

Hang Chu[1,2,4] Daiqing Li[4] David Acuna[1,2,4] Amlan Kar[1,2,4] Maria Shugrina[1,2,4] Xinkai Wei[1,4]
Ming-Yu Liu[4] Antonio Torralba[3] Sanja Fidler[1,2,4]
[1]University of Toronto [2]Vector Institute [3]MIT [4]NVIDIA

{chuhang1122,davidj,amlan}@cs.toronto.edu, {daiqingl,mshugrina,xinkaiw,mingyul,sfidler}@nvidia.com, torralba@mit.edu

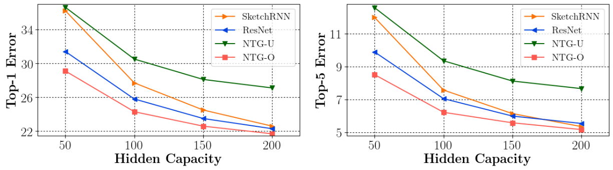


Figure 1: Top-1 error (left) and Top-5 error (right) on the QuickDraw [3] dataset, lower is better.

1. Learning Complex Shapes with Multi-Paths

To further investigate extracting complex shape representations using multiple paths similar to NTG, we take the QuickDraw [3] dataset. It contains 2.6 million sketch drawings of a wide range of 345 categories. We train models to extract features representing the sketch shape, which is then used recognizing the sketch's category.

We compare the following three methods:

- **SketchRNN [3]**: A sketch is treated as a single sequence and encoded with an RNN, which takes the relative motion as well as up-hold-down status of the pen as its inputs. We keep SketchRNN's encoder, and add a linear layer on top of the final representation for classification.
- **ResNet [4]**: We directly render the sketch on a 2D canvas and use an 18-layer CNN to form the representation vector, which is used to predict the category of the sketch.
- **NTG**: We treat the sketch as multiple paths, where each path represents a single stroke of pen being held down and drawing. Samilar to NTG, we use the sequence of relative motions between two nodes as input to the encoder. We compare the default unordered NTG (NTG-U), and an ordered version of NTG (NTG-O) where the hidden vector of each path is passed to another RNN according to their order in the drawing instead of direct summation in NTG-U.

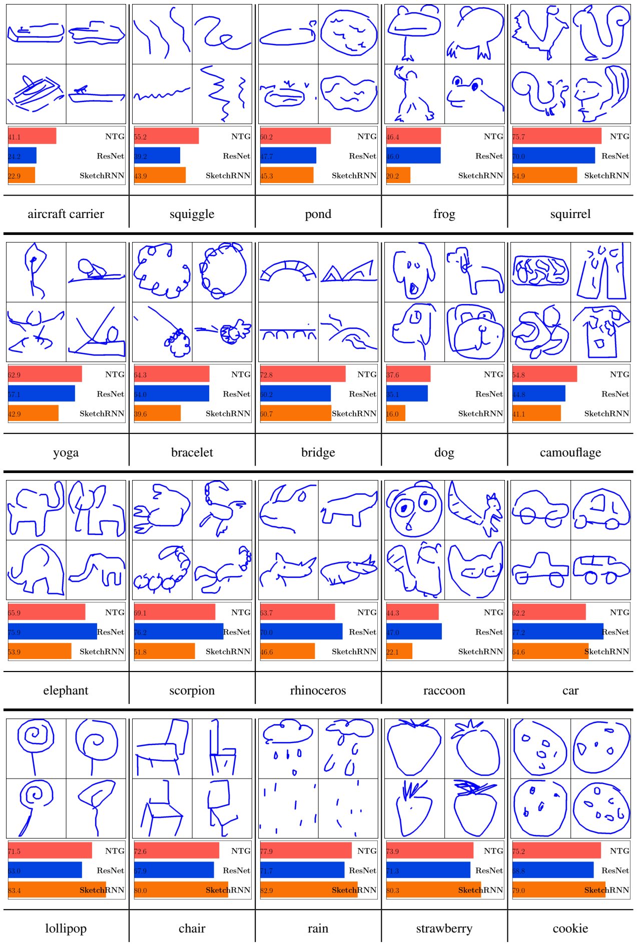


Figure 2: Examples of categories and their accuracies in the QuickDraw [3] dataset. First and second row show categories where NTG-O achieves better accuracy, third row and fourth row show categories where ResNet and SketchRNN achieve better accuracy, respectively. Although ResNet achieves good result, it uses 87% more number of weights than NTG. It can be seen that the multi-path NTG is particularly better at recognizing complex shapes such as aircraft carrier, camouflouge, and yoga.

For each method, we vary the dimension of the final representation vector from 50 to 200, and proportionally for the number of model weights where 200 corresponds to the full capacity. The result is shown in Fig. 1. It can be seen that NTG-U is able to achieve comparable accuracy, indicating that an effective representation of complex shape can be learned in an unordered manner from multiple paths of drawing motions. Stroke order seems arbitrary across

	mp1	mp2	pa	fc
NTG-Enhance	0.15	1.34	0.28	67.6
+ neighbours	0.29	2.44	0.53	85.6
- max-density	0.48	3.95	0.57	84.7
- max-degree	0.20	1.68	0.44	98.5
- planarity	0.95	7.29	0.70	75.7
- min-angle	0.23	1.93	0.47	79.4

Table 1: Ablation study on city generation perceptual FIDs.

different drawers. However, in a large scale dataset such as QuickDraw [3] order patterns emerge and become important to the success of this task, because the order is no longer unique and is shared by multiple drawers. Therefore, SketchRNN performs better than NTG-U as it is able to make use of stroke order. NTG-O achieves the best performance, because it not only learns path-wise features, but also makes use of stroke order information. It should be noted that ResNet18 has more than a magnitude more weights compared to SketchRNN and NTG. This shows the advantage of directly modeling the drawing motions, compared to learning from an image rendering. Fig. 2 shows qualitative examples. It can be seen that SketchRNN and ResNet are more effective in recognizing simple and regular shapes, while the multi-path NTG is better at recognizing complex classes, such as yoga, camouflage, and aircraft carrier.

2. City Road Layout Generation Details

We provide further details of our implementation in the first task of city road layout generation. In Fig. 3 and Fig. 4, we continue the Fig.3 of the main paper and provide more qualitative results.

• **GraphRNN-2D [7, 2]**: We combine the ideas from both GraphRNN [7] and RoadTracer [2]. Similar to GraphRNN, we generate new nodes in a BFS order, and use another RNN to determine edges connecting the new node to its 40 predecessors. Similar to RoadTracer [2], we render the local existing graph centered at the previous generated node and feed it as input when we predict the new node position. We further compute another mask that indicates bad positions which will cause invalid edge crossing to avoid placing the new node at these positions.

• **NTG**: At inference time, we find the maximum node degree and density plus the minimum angle between two edges that exist in the training set, and make sure NTG's generation does not exceed these limits. We enforce planarity by adding nodes when two edges intersect. In Tab. 1 we show ablation study of removing these constraints. We notice the max-density and max-degree constraints are useful in preventing over-generation. We also experiment adding neighbour nodes that are not connected by an immediate edge as context. This does not lead to improved performance as shown in Tab. 1. We conjecture this is due to the curriculum difference at inference time, where the neighbours grow in a directional manner.

	2	3	5	10	20	30
IOU	19.66	24.68	30.83	37.91	45.49	48.29
F1	32.86	39.59	47.12	54.98	62.53	65.13
APLS	45.77	43.97	44.16	42.68	43.99	42.94

Table 2: Effect of number of starting points in RoadTracer [2].

In NTG training, we sample incoming paths that have similar second last node in each iteration. This leads to better performance because it simulates the generation circumstance where active node often have one existing neighbour. Edges that have been produced by proximity check instead of being produced together with a decoded node, are not included in incoming paths of future steps. This is because their relative motion can be slightly different from the decoder's output, which causes covariance shift under the discrete coordinate setting.

3. Satellite Road Parsing Details

We provide further details of our implementation in the second task of satellite road parsing.

• **DeepRoadMapper [6]**: We use the open-source implementation provided by the authors of RoadTracer [2].

• **RoadTracer [2]**: We use the official implementation provided by the authors. We notice starting points are important for the performance of RoadTracer. Note that all starting points are selected from ground truth in our experiment, which gives RoadTracer additional information. Tab. 2 shows our experiment on the effect of starting points. It can be seen that both IOU and F1 increase as starting points increase, indicating more roads are detected pixel-wise. However, APLS decreases as starting points increase. This is because with more starting points, RoadTracer often produces more isolated sets of roads and fails to link them together. This leads to more invalid (infinite length) Dijkstra paths, thus causing lower APLS.

• **NTG**: In the parsing mode NTG-P, we discretize the decoder's polar coordinate output at a resolution of 10 degrees. Once an edge is generated, we compute its precise fine angle by applying threshold and thinning of the image probability within the range of the 10 degrees. In NTG-I, we start with the graph produced by threshold and thinning, and push nodes with one existing edge to the queue to start the NTG process.

4. Dataset, Code, and Video

Our data and code are available at https://nv-tlabs.github.io/NTG. We thank ESRI for letting us use the CityEngine software in demonstration video.

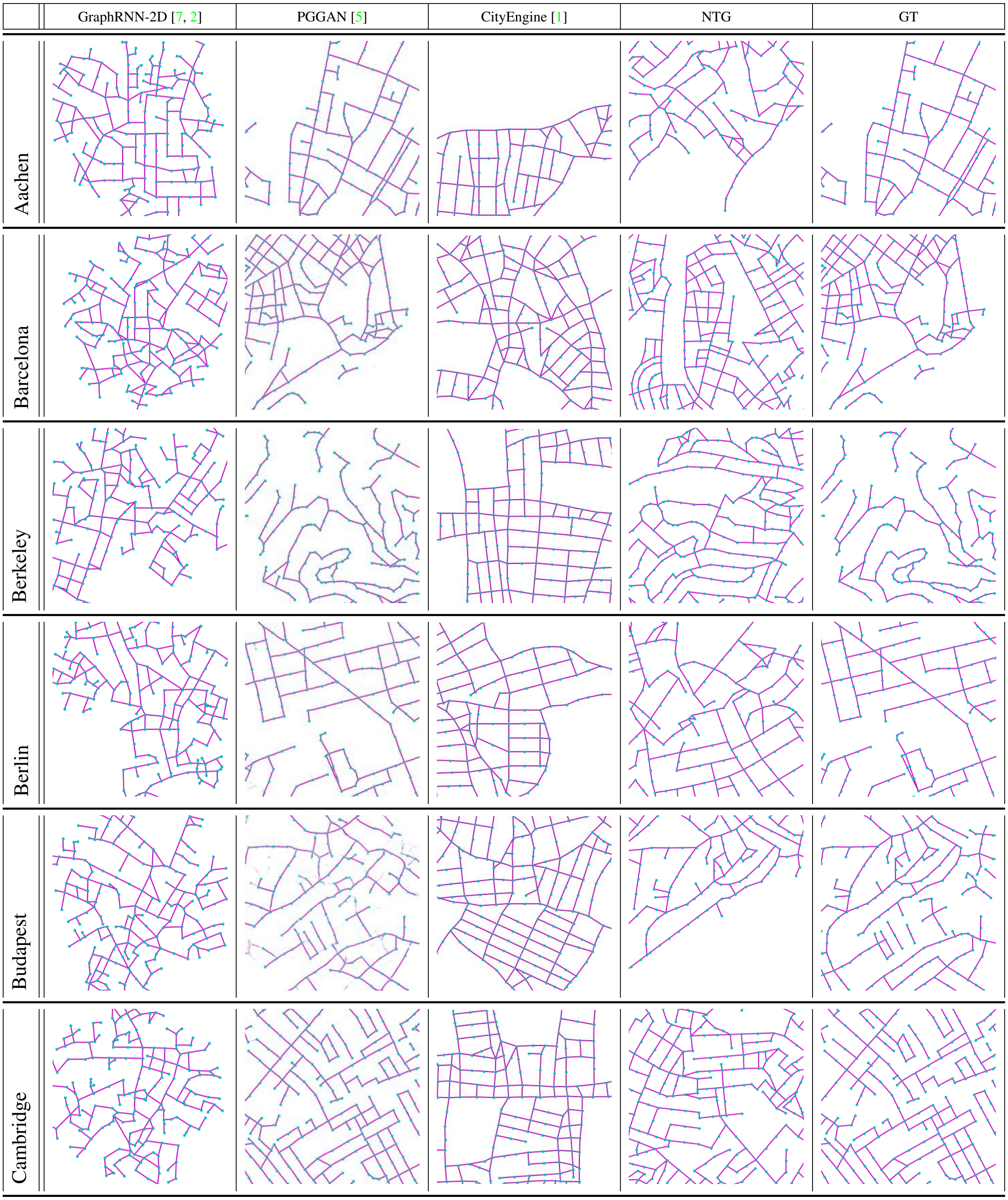


Figure 3: Main paper Fig.3 continued. More qualitative examples of city road layout generation.

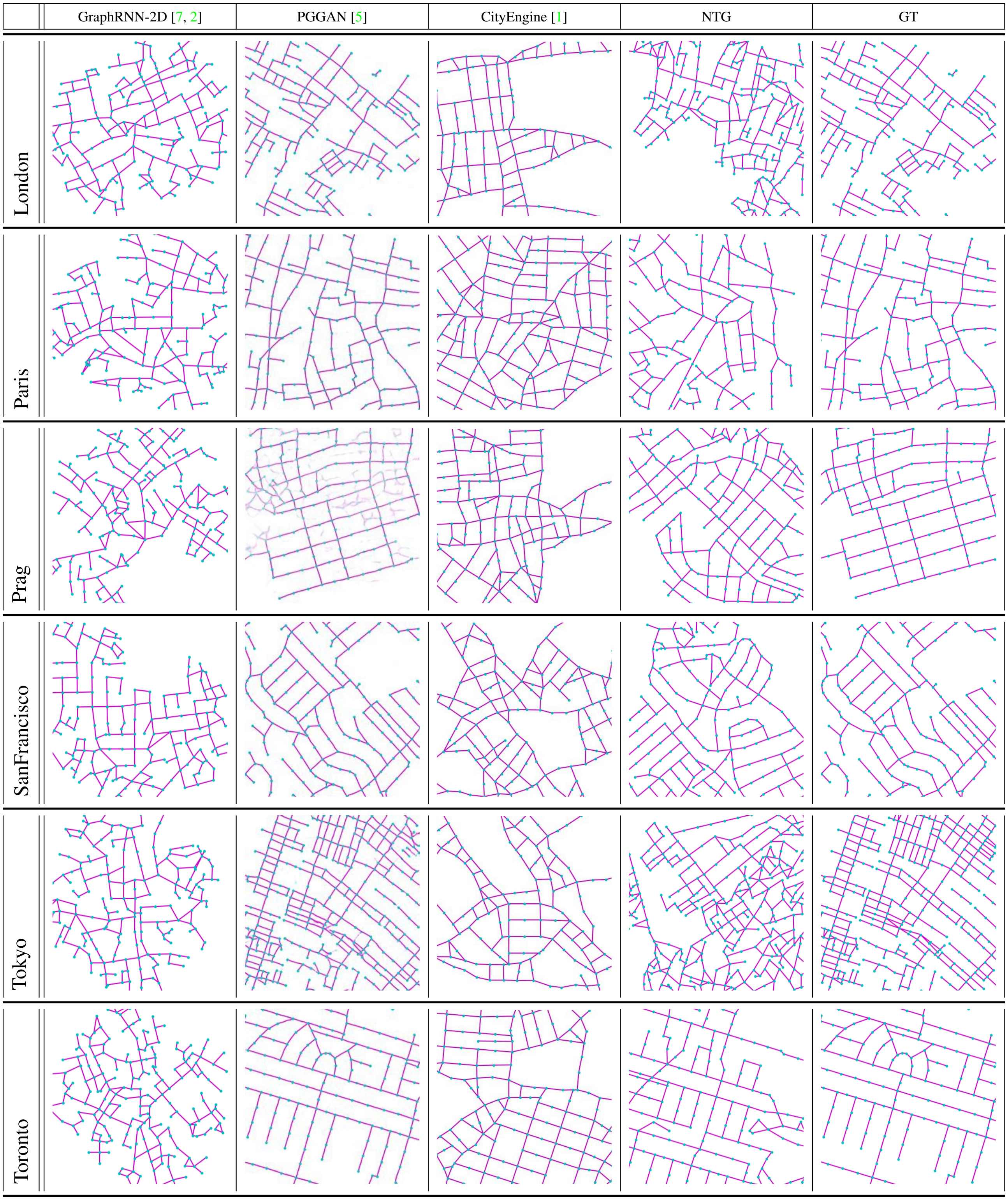


Figure 4: Main paper Fig.3 continued. More qualitative examples of city road layout generation.

References

[1] Esri: Cityengine. https://www.esri.com/en-us/arcgis/products/esri-cityengine. 3, 4

[2] Favyen Bastani, Songtao He, Sofiane Abbar, Mohammad Alizadeh, Hari Balakrishnan, Sanjay Chawla, Sam Madden, and David DeWitt. Roadtracer: Automatic extraction of road networks from aerial images. In *CVPR*, 2018. 2, 3, 4

[3] David Ha and Douglas Eck. A neural representation of sketch drawings. *arXiv:1704.03477*, 2017. 1, 2

[4] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016. 1

[5] Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen. Progressive growing of gans for improved quality, stability, and variation. In *ICLR*, 2018. 3, 4

[6] Gellért Máttyus, Wenjie Luo, and Raquel Urtasun. Deeproadmapper: Extracting road topology from aerial images. In *ICCV*, 2017. 2

[7] Jiaxuan You, Rex Ying, Xiang Ren, William Hamilton, and Jure Leskovec. Graphrnn: Generating realistic graphs with deep auto-regressive models. In *ICML*, 2018. 2, 3, 4