

Neural Turtle Graphics for Modeling City Road Layouts

Appendix

Hang Chu^{1,2,4} Daiqing Li⁴ David Acuna^{1,2,4} Amlan Kar^{1,2,4} Maria Shugrina^{1,2,4} Xinkai Wei^{1,4}
 Ming-Yu Liu⁴ Antonio Torralba³ Sanja Fidler^{1,2,4}
¹University of Toronto ²Vector Institute ³MIT ⁴NVIDIA

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

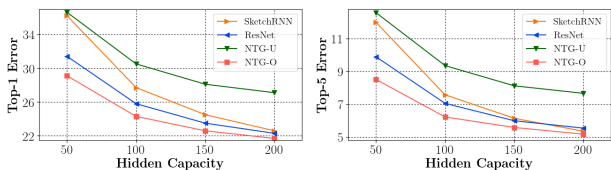


Figure 1: Top-1 error (left) and Top-5 error (right) on the QuickDraw [3] dataset, lower is better.

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. Similar 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.

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

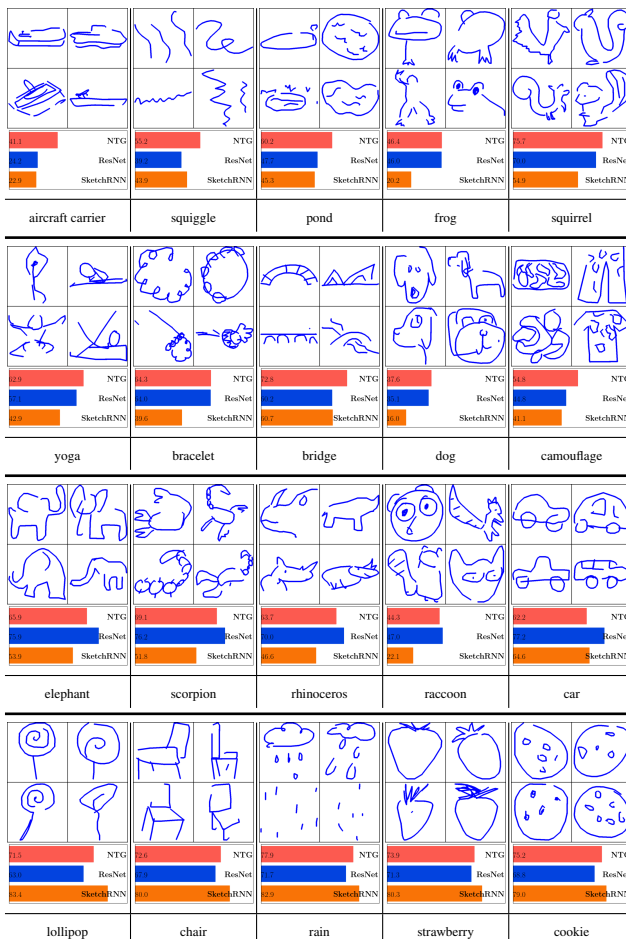


Figure 2: Examples of categories and their accuracies in the QuickDraw [3] dataset. First and second row show categories where NTG-O achieves better accuracy, third row and fourth row show categories where ResNet and SketchRNN achieve better accuracy, respectively. Although ResNet achieves good result, 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, camouflage, and yoga.

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

	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].

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.

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.

	GraphRNN-2D [7, 2]	PGGAN [5]	CityEngine [1]	NTG	GT
Aachen					
Barcelona					
Berkeley					
Berlin					
Budapest					
Cambridge					

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

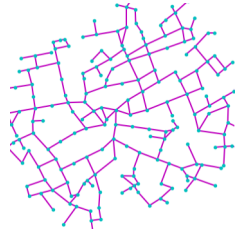
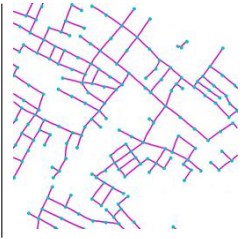
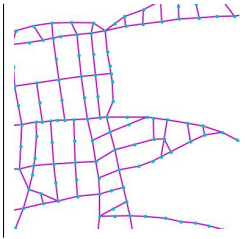
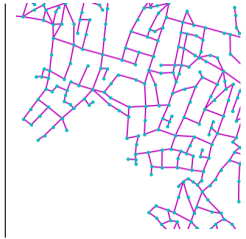

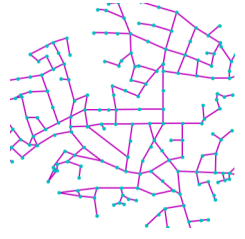
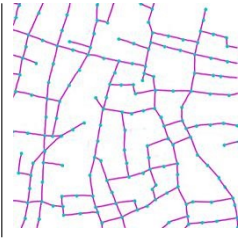
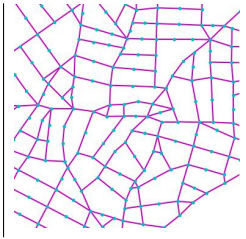
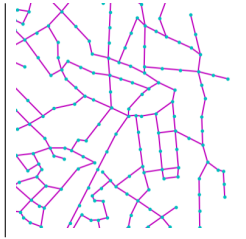
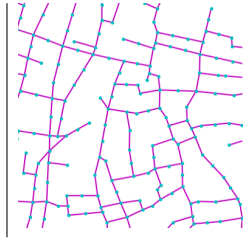
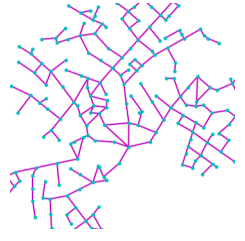

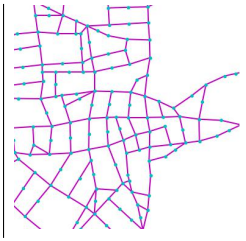

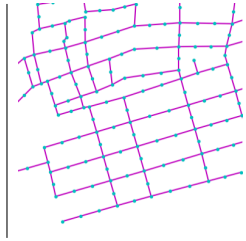
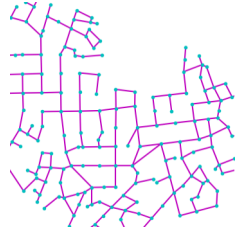
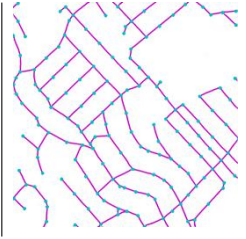
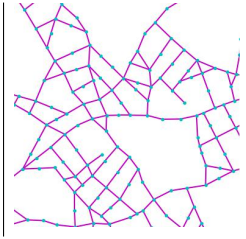

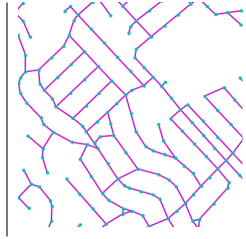
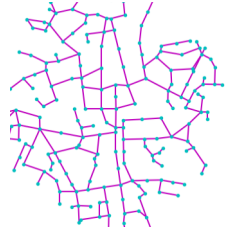
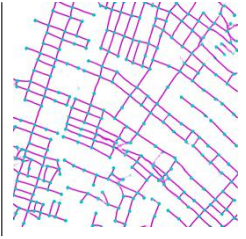
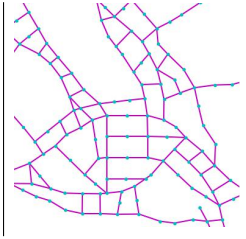
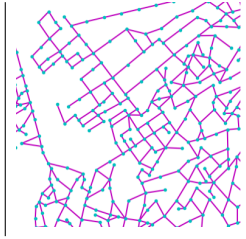
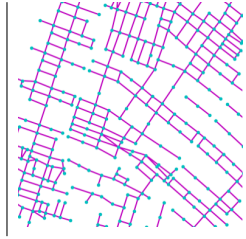
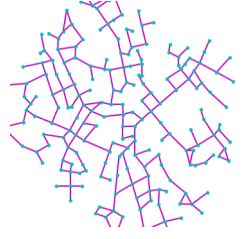
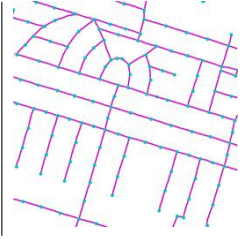
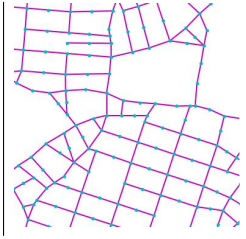
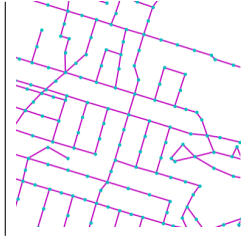
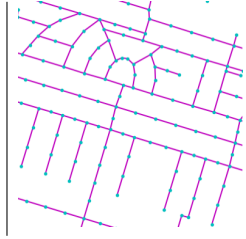
	GraphRNN-2D [7, 2]	PGGAN [5]	CityEngine [1]	NTG	GT
London					
Paris					
Prag					
SanFrancisco					
Tokyo					
Toronto					

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 Mátyus, Wenjie Luo, and Raquel Urtasun. Deep-roadmapper: 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