Supplementary Material: Learning to Assemble Geometric Shapes

Jinhwi Lee[1,2*], **Jungtaek Kim**[1*], **Hyunsoo Chung**[1], **Jaesik Park**[1] and **Minsu Cho**[1]
[1]Pohang University of Science and Technology (POSTECH)
[2]POSCO
{jinhwi, jtkim, hschung2, jaesik.park, mscho}@postech.ac.kr

In this material, we describe the detailed contents that supplement our main article.

S.1 Details of Shape Assembly

Algorithm s.1 Geometric Shape Assembly Procedure

Input: Fragments $\mathcal{X} = \{\mathbf{x}_i\}_{i=1}^{N}$ and a target shape S.
Output: Output shape $\mathbf{c}$.
1: Initialize a current shape $\mathbf{c}$ on a space Φ.
2: **while** $\mathcal{X} \neq \emptyset$ **do**
3: Select $\mathbf{x}_i \in \mathcal{X}$ and place it on Φ.
4: Assemble $\mathbf{c} \leftarrow$ shape $\mathbf{c}$ updated by the placement of $\mathbf{x}_i$.
5: Update $\mathcal{X} \leftarrow \mathcal{X} \setminus \mathbf{x}_i$.
6: **end while**
7: **return** $\mathbf{c}$

We provide the detailed procedure of shape assembly in Algorithm s.1.

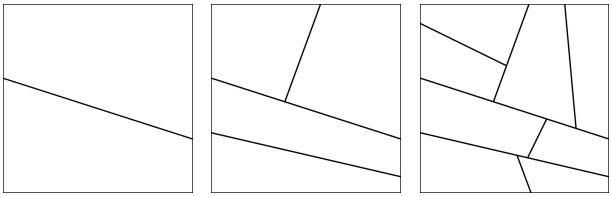
S.2 Details of Shape Fragmentation

We present the procedure of shape fragmentation in Algorithm s.2.

Algorithm s.2 Shape Fragmentation

Input: A target shape S, the number of partitions K.
Output: Fragments partitioned $\mathcal{X}_K = \{\mathbf{x}_i\}_{i=1}^{2^K}$.
1: Initialize a set of fragments $\mathcal{X}_0 = \{S\}$.
2: **for** $i = 1, \ldots, K$ **do**
3: Initialize $\mathcal{X}_i = \{\}$.
4: **for all** $\mathbf{x} \in \mathcal{X}_{i-1}$ **do**
5: Partition a fragment $\mathbf{x}$ to $\mathbf{x}_+, \mathbf{x}_-$.
6: Update $\mathcal{X}_i \leftarrow \mathcal{X}_i + \{\mathbf{x}_+, \mathbf{x}_-\}$.
7: **end for**
8: **end for**
9: Rotate all the fragments in $\mathcal{X}_K$ at random.
10: **return** a set of fragments $\mathcal{X}_K$

We divide a target geometric shape into fragments using a binary space partitioning algorithm [Schumacher, 1969]. The number of fragments increases exponentially with the number of partitions. Some of partitioning examples are shown in Figure s.1.



(a) Partitioning 1 (b) Partitioning 2 (c) Partitioning 3

Figure s.1: Fragmentation examples on `Square` by the number of partitions.

We create a shape fragmentation dataset under two constraints that prevent generating small fragments. First, the hyperplane does not cross adjacent edges of the given polygon. Second, the position that the hyperplane will pass through is randomly selected between the range of $\pm$ 25% of edge length from the center of the edge. The dataset we use is composed of 5,000 samples.

Sequence Sampling Strategy. As presented in Table 7, the consistency of assembling sequences in the dataset turns out to be crucial in training our network. If we employ the assembly order randomly determined in training our network, it performs significantly worse. To achieve better performance, we should select the next fragment that is adjacent to the fragments already chosen. As inspired by human assembling process [Chazelle, 1983], we determine the order of fragments from bottom, left to top, right by considering the center position of fragments in the target shape.

S.3 Details of Experiments

Our framework is implemented using TensorFlow [Abadi *et al.*, 2016]. We conduct the experiments on a Ubuntu 18.04 workstation with twelve Intel(R) Core i7-5930K CPU running at 3.50 GHz, 64 GB random access memory and NVIDIA Titan X (pascal) GPU.

To train our model with the objectives defined in the main article, we need to balance the terms in two objectives. Thus, we do our best to find the coefficients for these objectives. As a result, we scale up the term for determining the position

*Equal contribution.

Network	Layer	Output Dimension
Generator	Reshape	64
	FC	128
	BatchNorm	128
	ReLU	128
	FC	64
	BatchNorm	64
	ReLU	64
	FC	32
	BatchNorm	32
	ReLU	32
	FC	16
Discriminator	Reshape	64
	FC	128
	ReLU	128
	Dropout 0.3	128
	FC	64
	ReLU	64
	Dropout 0.3	64
	FC	32
	ReLU	32
	Dropout 0.3	32
	FC	16

Table s.1: Architecture of V-GAN.

of the next fragment by multiplying 1,000 and the term for a rotation probability by multiplying 10.

S.3.1 Baselines

We describe the details of baselines we employ in this paper.

Simulated annealing (SA). Similar to our proposed method, SA [Pincus, 1970] iteratively selects and places one fragment such that IoU between the target object and assembled fragments is maximized. Thus, the optimization is repeated until the number of assembled fragments is equal to the given total budget.

Bayesian optimization (BayesOpt). BayesOpt [Brochu *et al.*, 2010] is used to find the pose of given fragments by maximizing IoU between fragments and a target object. Similar to SA, the optimization is repeated in total of the number of fragments. We utilize the Bayesian optimization package [Kim and Choi, 2017] to implement this baseline.

LayoutGAN modified for vertex inputs (V-GAN). It is a modification of [Li *et al.*, 2020]. Since LayoutGAN assumes that fragment shapes are always fixed and only finds their positions, we modify the generator structure to a neural network that takes fragment vertices as inputs and predicts center coordinates as their positions. Discriminator then makes real or fake decision based on adjusted vertex coordinates. Similar to GAN [Goodfellow *et al.*, 2014], it is trained via an adversarial training strategy.

S.3.2 Model Architecture

We introduce the model architecture for FAN including FRAM and V-GAN.

FAN. FAN consists of FAN-Select that selects next fragment and FAN-Pose that predicts center coordinates of the chosen fragment. There exist 7 hyperparameters in FAN with 3 specifically in FRAM. Full details of the pipeline can be found in Table s.2. The default dimension of fully connected layer in the whole pipeline is fixed to 256 throughout the experiments. For the training, we use batch size of 32 and Adam optimizer [Kingma and Ba, 2015] for both FAN-Select and FAN-Pose with learning rate of 1×10^{-3} respectively. FAN-Select is a convolutional neural network (CNN) followed by FRAM including MLP block. FAN-Pose is convolutional encoder-decoder with MLP block followed by an additional fully-connected layer inserted in between. Each encoder for FAN-Pose and FAN-Select do not share the weights, and thus are trained with different loss. Our FRAM follows a similar setting of the original Transformer model [Vaswani *et al.*, 2017] with 2,048 hidden units and 8 attention heads.

V-GAN. This model follows typical setup of generative adversarial network [Goodfellow *et al.*, 2014], having both generator and discriminator. Specifically, a generator takes fragment's vertex coordinates as inputs and predicts center coordinates. Then fragment's vertex coordinates are translated by predicted center coordinates. Next, a discriminator makes fake or real decision based on translated vertex coordinates. For training, we use batch size of 128 and Adam optimizer with learning rate of 1×10^{-4}. The details of model is summarized in Table s.1.

S.4 Additional Discussion

Our proposed network can be viewed as a permutation-equivariant network, which satisfies $f(\pi([\mathbf{x}_1, \ldots, \mathbf{x}_n]^\top)) = \pi[f([\mathbf{x}_1, \ldots, \mathbf{x}_n]^\top)]$, where $f : \mathbb{R}^{n \times d} \rightarrow \mathbb{R}^{n \times d'}$ is a neural network that outputs a set and π is a permutation function along the first dimension. The work [Guttenberg *et al.*, 2016] suggests a permutational layer to handle the different inputs and does not depend on the specific order of inputs. The work of [Zaheer *et al.*, 2017] derives the necessary and sufficient conditions on permutation-equivariant networks. Moreover, an attention-based block for sets has been proposed [Lee *et al.*, 2019].

Furthermore, the number of learnable parameters does not depend on the dimensionality of $\mathbf{y}$ and the sizes of $\mathbf{M}_i$ and $\mathbf{r}_i$ for $i \in \{1, \ldots, |\mathcal{X}|\}$. As shown in Eq. (1) and Eq. (2), their outputs can handle variable-sized $\mathcal{X}$ and the order of $\mathcal{X}$ affects the order of the outputs, satisfying the permutation equivariance. The experimental results support that our method can learn these challenging scenarios.

References

[Abadi *et al.*, 2016] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, et al. TensorFlow: A system for large-scale machine learning. In *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, pages 265–283, Savannah, Georgia, USA, 2016.

[Brochu *et al.*, 2010] E. Brochu, V. M. Cora, and N. de Freitas. A tutorial on Bayesian optimization of expensive

Network	Layer	Output Dimension
Encoder	Conv 2D, 16 ch., 3×3, stride 1, padding 1	$128 \times 128 \times 16$
	ReLU	$128 \times 128 \times 16$
	Dropout 0.1	$128 \times 128 \times 16$
	Conv 2D, 16 ch., 3×3, stride 1, padding 1	$128 \times 128 \times 16$
	ReLU	$128 \times 128 \times 16$
	Maxpool 2D, 2×2, stride 2, padding 0	$64 \times 64 \times 16$
	Conv 2D, 32 ch., 3×3, stride 1, padding 1	$64 \times 64 \times 32$
	ReLU	$64 \times 64 \times 32$
	Dropout 0.1	$64 \times 64 \times 32$
	Conv 2D, 32 ch., 3×3, stride 1, padding 1	$64 \times 64 \times 32$
	ReLU	$64 \times 64 \times 32$
	Maxpool 2D, 2×2, stride 2, padding 0	$32 \times 32 \times 32$
	Conv 2D, 64 ch., 3×3, stride 1, padding 1	$32 \times 32 \times 64$
	ReLU	$32 \times 32 \times 64$
	Dropout 0.2	$64 \times 64 \times 32$
	Conv 2D, 64 ch., 3×3, stride 1, padding 1	$32 \times 32 \times 64$
	ReLU	$32 \times 32 \times 64$
	Flatten	65,536
	Linear	256
	ReLU	256
Decoder	Linear	32,768
	ReLU	32,768
	Reshape	$32 \times 32 \times 32$
	ConvTranspose 2D, 32 ch., 2×2, stride 2, padding 2	$64 \times 64 \times 32$
	Conv 2D, 32 ch., 3×3, stride 1, padding 1	$64 \times 64 \times 32$
	ReLU	$64 \times 64 \times 32$
	Dropout 0.1	$64 \times 64 \times 32$
	Conv 2D, 32 ch., 3×3, stride 1, padding 1	$64 \times 64 \times 32$
	ReLU	$64 \times 64 \times 32$
	ConvTranspose 2D, 16 ch., 2×2, stride 2, padding 2	$128 \times 128 \times 32$
	Conv 2D, 32 ch., 3×3, stride 1, padding 1	$128 \times 128 \times 16$
	ReLU	$128 \times 128 \times 16$
	Dropout 0.1	$128 \times 128 \times 16$
	Conv 2D, 32 ch., 3×3, stride 1, padding 1	$128 \times 128 \times 16$
	ReLU	$128 \times 128 \times 16$
	Conv 2D, 1 ch., 1×1, stride 1, padding 0	$128 \times 128 \times 1$
MLP	Linear	256
	Batch normalization	256
	ReLU	256
	Linear	256
	Batch normalization	256
	ReLU	256
	Linear	1 or 256
FRAM		
FRAM-Encoder	Encoder layer $\times$ 2 (including MHA. $\times$ 8)	256
FRAM-Decoder	Decoder layer $\times$ 2 (including MHA. $\times$ 8)	256

Table s.2: Architecture of FAN.

cost functions, with application to active user modeling and hierarchical reinforcement learning. *arXiv preprint arXiv:1012.2599*, 2010.

[Chazelle, 1983] B. Chazelle. The bottomn-left bin-packing heuristic: An efficient implementation. *IEEE Transactions on Computers*, C-32(8):697–707, 1983.

[Goodfellow *et al.*, 2014] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 27, pages 2672–2680, Montreal, Quebec, Canada, 2014.

[Guttenberg *et al.*, 2016] N. Guttenberg, N. Virgo, O. Witkowski, H. Aoki, and R. Kanai. Permutation-equivariant neural networks applied to dynamics prediction. *arXiv preprint arXiv:1612.04530*, 2016.

[Kim and Choi, 2017] J. Kim and S. Choi. BayesO: A Bayesian optimization framework in Python. https://bayeso.org, 2017.

[Kingma and Ba, 2015] D. P. Kingma and J. L. Ba. Adam: A method for stochastic optimization. In *Proceedings of the International Conference on Learning Representations (ICLR)*, San Diego, California, USA, 2015.

[Lee *et al.*, 2019] J. Lee, Y. Lee, J. Kim, A. R. Kosiorek, S. Choi, and Y. W. Teh. Set Transformer: A framework for attention-based permutation-invariant neural networks. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 3744–3753, Long Beach, California, USA, 2019.

[Li *et al.*, 2020] J. Li, J. Yang, A. Hertzmann, J. Zhang, and T. Xu. LayoutGAN: Synthesizing graphic layouts with vector-wireframe adversarial networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2020.

[Pincus, 1970] M. Pincus. Letter to the editor – a Monte Carlo method for the approximate solution of certain types of constrained optimization problems. *Operations Research*, 18(6):1225–1228, 1970.

[Schumacher, 1969] R. Schumacher. Study for applying computer-generated images to visual simulation. Technical Report AFHRL-TR-69-14, Air Force Human Resources Laboratory, Air Force Systems Command, 1969.

[Vaswani *et al.*, 2017] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 30, pages 5998–6008, Long Beach, California, USA, 2017.

[Zaheer *et al.*, 2017] M. Zaheer, S. Kottur, S. Ravanbakhsh, B. Poczos, R. R. Salakhutdinov, and A. J. Smola. Deep sets. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 30, pages 3391–3401, Long Beach, California, USA, 2017.