SuperSVG: Superpixel-based Scalable Vector Graphics Synthesis (Supplementary Material)

Teng Hu[1], Ran Yi[1]*, Baihong Qian[1], Jiangning Zhang[2], Paul L. Rosin[3], Yu-Kun Lai[3]
[1]Shanghai Jiao Tong University, [2]Youtu Lab, Tencent, [3]Cardiff University
{hu-teng, ranyi, cherry_qbh}@sjtu.edu.cn, vtzhang@tencent.com, {RosinPL, LaiY4}@cardiff.ac.uk

1. Overview

In this supplementary material, more details about the proposed SuperSVG method and more experimental results are provided, including:

- More details about our Dynamic Path Warping (DPW) (Section 2);
- More details for the experiments (Section 3);
- Comparison experiments on Emoji dataset (Section 4);
- More ablation studies (Section 5);
- Additional comparison experiments (Section 6);
- More results of our method (Section 7).

2. Details about Our Dynamic Path Warping

Problem Insight. Given the generated path sequence $S' = \{s'_1, s'_2 \cdots s'_m\}$ and the target path sequence $S = \{s_1, s_2 \cdots s_n\}$, we aim to find the distance between the two path sequences in path parameter space. Denote the distance matrix between each path pair of S and S' as $D = \{d_{i,j}\}_{n\times m}$, with $d_{i,j}$ being the distance between s_i and s'_j. Our DPW aims to find an optimal matching function $match(j)$ that minimizes the objective function:

$$\sum_{j=1}^{m} \|d_{match(j),j}\|, \tag{1}$$

where $match(j) \geq match(j-1)$[1].

We transform the problem into finding an optimal path in a Cartesian grid. In Fig. 1, the distance matrix can be represented as an $m\times n$ grid, where the yellow point corresponds to one mapping of $match(j) = i$, while the black point indicates non-matching (*i.e.,* not added in the objective function), with the yellow point denoted as $(i, j, 1)$ and the black point denoted as $(i, j, 0)$. Therefore, any matching function $match(j)$, where $1 \leq j \leq m$, can be represented by m yellow points in the grid, with each yellow point in a different row. Considering every two adjacent rows have two yellow points $(match(j-1), j-1, 1)$ and $(match(j), j, 1)$, with $match(j-1) \leq match(j)$, by adding black points between the two yellow points when they are not adjacent, the m yellow points and the added black points can form a path in the grid that starts from the bottom left corner to the top right corner, which only consists of rightward, upward, and diagonal up-right movements, and can be further computed by dynamic programming. In this way, the problem of finding an optimal matching function is transformed to finding an optimal path in the distance grid.

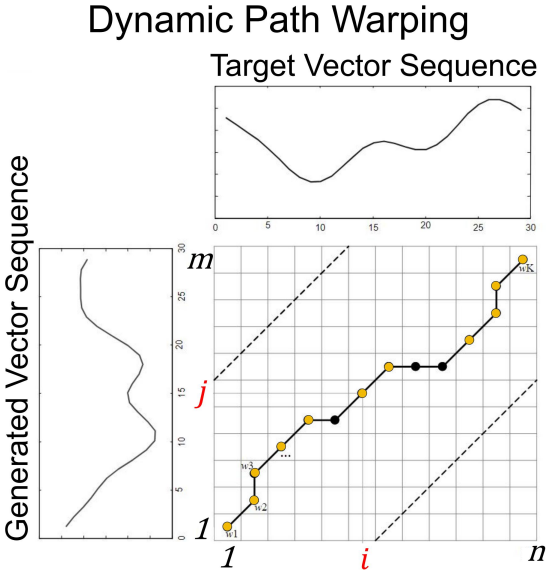

Figure 1. Illustration of our Dynamic Path Warping. Given a distance matrix $D_{n\times m}$ (with $d_{i,j}$ being the distance between s_i and s'_j) represented as a grid, our DPW aims to find an optimal grid path composed of yellow and black points, to minimize the objective function (Eq.(1)). The yellow point $(i, j, 1)$ represents a matching $match(j) = i$, and the black point $(i, j, 0)$ represents non-matching $match(j) \neq i$ (*i.e.,* it is not counted in the objective function).

Specifically, the path should satisfy the following conditions:

1) For each row of the path, there should be one and only one yellow point, i.e., there will not be two adjacent yellow points in the same row (with the same j).[2]

2) The path contains both the yellow points $(i, j, 1)$ that represent $match(j) = i$ and black points $(i, j, 0)$ that represent $match(j) \neq i$.

3) For each point in the path, it can only move to the right, upward, or diagonally upward to the right.

It can be easily seen that the final value of DPW only depends on the yellow points (matching points), while the values of black points (non-matching points) are not

*Corresponding author.

[1]Each generated path s'_j, should be matched to one and only one target path s_i but a target path s_i may be unmatched to any generated path s'_j.

[2]The underlying reason is that we require one generated path to only correspond to one target path, in order to avoid a single generated path being the average of several target paths.

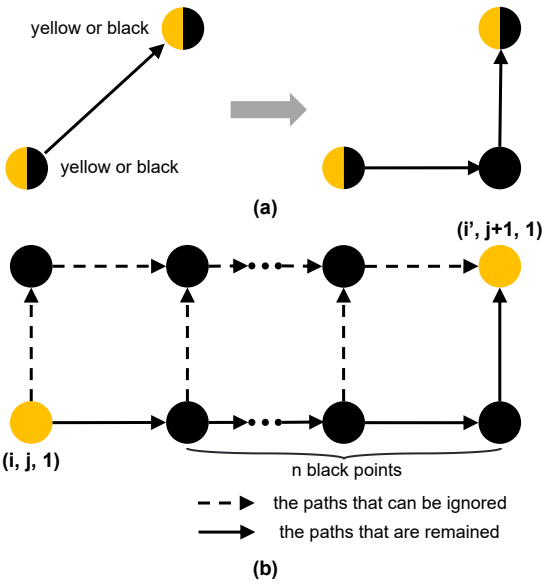

Figure 2. Path simplification for DPW. (a) **Property 1**: The movement from (i, j) to $(i+1, j+1)$ can be simplified to moving from (i, j) rightward to $(i + 1, j, 0)$ (black point) and then moving upward to $(i + 1, j + 1)$. (b) **Property 2**: Although there are multiple paths (composed of n black points) between two yellow points $(i, j, 1)$ and $(i', j + 1, 1)$ in adjacent rows, they are equivalent in terms of their objective function values. Therefore, we can ignore the transition modes represented by the dashed arrows, and only need to consider the solid arrows.

counted in the objective function. Then, we can simplify the form of the path for DPW based on the following properties.

Property 1. *We can ignore movements that are diagonally upwards to the right (i.e., from (i, j) to $(i+1, j+1)$).*

Explanation 1. *For any diagonal up-right movement from colored point (i, j) (black or yellow) to point $(i + 1, j + 1)$ (Fig. 2(a)), it is equal to first moving to the right to the black point $(i + 1, j, 0)$ and then moving upwards to $(i + 1, j + 1)$: By adding the intermediate black point $(i + 1, j, 0)$, 1) the colors of the two endpoints (i, j), and $(i + 1, j + 1)$ are not changed, and 2) the intermediate black point $(i + 1, j, 0)$ does not contribute to the value of the DPW objective function. Therefore, we can consider only rightward and upward movements.*

Property 2. *We can consider only four possible state transition modes: $(i, j, 1) \rightarrow (i, j + 1, 1)$, $(i, j, 1) \rightarrow (i + 1, j, 0)$, $(i, j, 0) \rightarrow (i+1, j, 0)$, and $(i, j, 0) \rightarrow (i, j+1, 1)$.*

Explanation 2. *For two yellow points $(i, j, 1)$ and $(i', j + 1, 1)$ in adjacent rows, their positional relationship can be summarized into Fig. 2(b), where the number of black points n between them is larger than or equal to 0[3]. When $n = 0$, $(i, j, 1)$ just moves upward to $(i, j+1, 1)$. When $n > 0$, the paths between $(i, j, 1)$ and $(i', j + 1, 1)$ can be any composition of the dashed and solid arrows that moves from $(i, j, 1)$ to $(i', j + 1, 1)$ in Fig. 2(b), i.e., there are different ways to add black points between the two yellow points to connect into a path. However, all the possible paths contribute to the same result since the value of the DPW objective function only depends on the yellow points. Considering the different paths between these two yellow points are equal in objective function values, we can simplify the paths by ignoring the transition modes represented by the dashed arrows, and only need to consider the transition denoted by solid arrows. Therefore, there are only 4 remaining state transition modes (Fig. 2(b)):*

(1) $(i, j, 1) \rightarrow (i, j + 1, 1)$ (yellow upward to yellow),
(2) $(i, j, 1) \rightarrow (i + 1, j, 0)$ (yellow rightward to black),
(3) $(i, j, 0) \rightarrow (i + 1, j, 0)$ (black rightward to black),
(4) $(i, j, 0) \rightarrow (i, j + 1, 1)$ (black upward to yellow).

[3]Since here $i = match(j) \geq match(j - 1) = i'$.

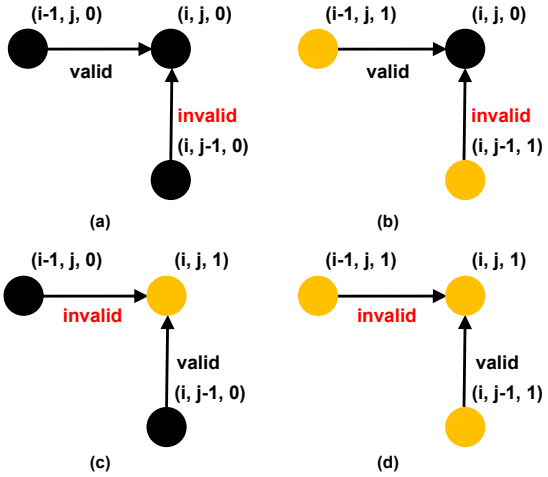

Figure 3. The illustration of valid paths and invalid paths after path simplification for DPW. There are 8 transition modes in total. But according to the *Properties 1 and 2*, half of the modes can be ignored (set as invalid), which greatly simplifies the dynamic programming process.

Dynamic Programming Solution. With the simplified state transition modes, we can compute our DPW by dynamic programming. Specifically, we define $p_{i,j}$ as the minimum accumulated distance when going from the start point $(1, 1)$ to the yellow (matching) point $(i, j, 1)$, and $q_{i,j}$ as the minimum accumulated distance when going from the start point $(1, 1)$ to the black (non-matching) point $(i, j, 0)$. According to *Properties 1 and 2*, there are only 4 state transition modes:

$$\begin{aligned} &(1) p_{i,j-1} \rightarrow p_{i,j}, \\ &(2) p_{i-1,j} \rightarrow q_{i,j}, \\ &(3) q_{i-1,j} \rightarrow q_{i,j}, \\ &(4) q_{i,j-1} \rightarrow p_{i,j}, \end{aligned} \tag{2}$$

which are shown in Fig. 3 (yellow for p and black for q).

Therefore, we have the state transition function:

$$\begin{aligned} p_{i,j} &= d_{i,j} + \overset{\gamma}{\min}(q_{i,j-1}, p_{i,j-1}), \\ q_{i,j} &= \overset{\gamma}{\min}(q_{i-1,j}, p_{i-1,j}), \end{aligned} \tag{3}$$

where $d_{i,j}$ is the distance between path s_i and path s'_j, and the final value of our DPW objective function is $\min^{\gamma}(p_{n,m}, q_{n,m})$.

Overall, our DPW can be regarded as an extended version of DTW [8], with our objective function specifically designed for measuring the distance between two SVG path sequences. Although there are other versions of DTW designed for different problems, to the best of

our knowledge, our DPW is the first differentiable version that aims at the specially designed objective of alignment of SVG path sequences.

3. More Details for the Experiments

More Implementation Details. In the experiments of the main paper, we have evaluated the performance of SuperSVG under different numbers of paths. For a certain number of paths n, we assign about half of the paths to the coarse-stage model and half to the refinement-stage model. Specifically, with our path efficiency loss $\mathcal{L}_{PE}$, our coarse-stage model predicts around 32 visible paths for a superpixel on average. Therefore, we decompose the target image into $n_1 = \frac{n}{2\times32}$ superpixels and employ the coarse-stage model to predict SVG paths for each of them. We combine all the visible paths output from the coarse-stage model, and employ the refinement-stage model to add more paths onto each superpixel repeatedly until the total path number reaches n.

Evaluation Metrics. In the experiments, we use four metrics to evaluate the vectorization results, comparing the rendered image of the output SVG to the target image:

(1) MSE Distance: Mean Squared Error (MSE) is a widely used metric in image processing to assess the quality of image reconstruction. It measures the average squared difference between the original and reconstructed images, with lower MSE values indicating better image fidelity.

(2) PSNR: The Peak Signal-to-Noise Ratio (PSNR) is one of the most prevalent and extensively utilized metrics for assessing image quality. A higher PSNR value indicates a superior quality of image reconstruction.

(3) LPIPS: The Learned Perceptual Image Patch Similarity (LPIPS) [18] is a perceptual metric utilized for assessing the similarity between two images. A lower LPIPS value indicates a higher similarity between the output image and the target image.

(4) SSIM: Structure Similarity Index Measure (SSIM) [16] is derived from three aspects of image similarity: luminance, contrast and structure, based on the idea that the pixels have strong inter-dependencies especially when they are spatially close. The higher the SSIM score is, the more similar the two images are.

Network Architecture. Our **Coarse-stage model** consists of three modules: one vision transformer encoder; one cross-attention module; and one self-attention module. **1)** The vision transformer encoder [9] employs the ViT implementation from PyTorch Image Models (timm) [5], which takes an 224×224 image as input and splits the image into patches (tokens) with size 16×16. **2)** The cross-attention module takes the encoded feature as the Key and Value, and takes the learnable path queries as the Query. Then the cross-attention module is followed by a two-layer MLP with GELU activation. **3)** Moreover, the self-attention module is employed to further process the output from the cross-attention layer to project the image features into path parameters. And the self-attention module is also followed by a two-layer MLP with GELU activation.

Our **Refinement-stage model** first employs a three-layer convolution network with 3×3 convolution kernels to encode the current canvas and target superpixel into a $3 \times 224 \times 224$ feature map. And then it employs the same network as the coarse-stage model to project the image features into 128×27-dimension output. At last, a fully connected layer is employed to map it into path parameters with 8×27 dimension. *For more details, please refer to the code provided in the supplementary material.*

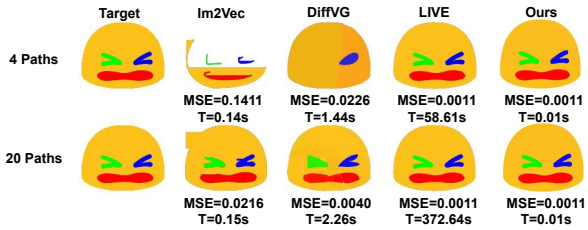

Figure 4. Comparison on EMOJIs [3]. We achieve the best reconstruction accuracy and highest speed (paths number 4 and 20).

Table 1. Ablation study on different superpixel methods.

Method	MSE ↓	PSNR ↑	LPIPS ↓	SSIM ↑
LSC [11]	0.0148	20.25	0.4631	0.6952
SEEDS [15]	0.0042	24.65	0.4333	0.7777
SpixelFCN [17]	0.0050	23.81	0.4576	0.7562
SNIC [6]	0.0038	25.26	0.4125	0.7997
SLIC-compact=10	0.0050	24.15	0.4388	0.7650
SLIC-compact=20	0.0040	25.02	0.4205	0.7900
SLIC-compact=30 (**Ours**)	**0.0032**	**26.04**	**0.4075**	**0.8111**

4. Comparison on Emoji

Since Im2Vec [13] is domain-specific and struggles to vectorize complex images, we compare with it on EMOJIS dataset [3] using its pretrained model. As shown in Fig. 4, our SuperSVG-B achieves the best reconstruction accuracy and highest speed.

5. More Ablation Studies

Ablation on Different Superpixel Methods. We conduct ablation studies on using different superpixel methods: we compare with using the representative superpixel methods including LSC [11], SEEDS [15], SpixelFCN [17] and SNIC [6], as well as SLIC [7] with different compactness (10, 20 and 30), to decompose the target image into superpixels, and then vectorize each superpixel separately. The comparison results with 1,000 SVG paths are shown in Table 1. It can be seen that SLIC works better with our proposed SVG synthesis framework, and a higher compactness in SLIC results in a better performance.

Ablation on Self-attention Module. We further conduct ablation studies on the number of self-attention modules in both our coarse-stage and refinement-stage models. In our model design, we employ one self-attention module each in coarse-stage and refinement-stage models. Then, we train 3 additional versions for each of the coarse-stage and refinement-stage models with 2, 4 and 8 self-attention modules respectively. The comparison results with 1,000 SVG paths are shown in Table 2. It can be seen that, as the number of the self-attention modules increases, the performance of the model does not have an obvious improvement. Therefore, we only employ one self-attention mod-

Table 2. Ablation study on the number of self-attention modules.

Coarse-stage Model	MSE ↓	PSNR ↑	LPIPS ↓	SSIM ↑
Self-attn×1 (**Ours**)	**0.0032**	**26.04**	**0.4075**	**0.8111**
Self-attn×2	**0.0032**	26.00	0.4079	0.8109
Self-attn×4	0.0034	25.80	0.4146	0.8081
Self-attn×8	0.0033	25.98	0.4080	0.8102

(a) Ablation on the number of self-attention modules in the coarse-stage model.

Refinement-stage Model	MSE ↓	PSNR ↑	LPIPS ↓	SSIM ↑
Self-attn×1 (**Ours**)	**0.0032**	**26.04**	**0.4075**	**0.8111**
Self-attn×2	0.0034	25.78	0.4135	0.8072
Self-attn×4	0.0033	25.89	0.4092	0.8096
Self-attn×8	**0.0032**	26.01	0.4080	0.8108

(b) Ablation on the number of self-attention modules in the refinement-stage model.

ule to achieve a good vectorization quality while keeping a higher efficiency and fewer learnable parameter numbers.

6. Additional Comparison Experiments

In this section, we show more comparison results with the state-of-the-art vectorization methods, LIVE [12], DiffVG [10], Adobe [1] and Potrace [14] under 500, 2,000 and 4,000 SVG paths. The results are shown in Figures 5–7. It can be seen that under the same number of SVG paths, our SuperSVG can reconstruct more details than the other methods in both the foreground and the background regions.

7. More Results of Our Method

To show the effectiveness of our model, we show more experimental results on high-resolution in-the-wild data collected from the Internet [2, 4]. We vectorize all the test images into 4,000 SVG paths using our SuperSVG-B and SuperSVG-F, and the results are shown in Figures 8–11. It can be seen that our SuperSVG achieves a good vectorization quality with rich details.

References

[1] Adobe Illustrator. https://www.adobe.com/products/illustrator.html. 4

[2] Free-Images. https://free-images.com/. 4

[3] Noto Emoji. https://github.com/googlefonts/noto-emoji. 3

[4] Pixabay. https://pixabay.com/. 4

[5] PyTorch Image Models. https://github.com/huggingface/pytorch-image-models. 3

[6] Radhakrishna Achanta and Sabine Susstrunk. Superpixels and polygons using simple non-iterative clustering. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4651–4660, 2017. 3

[7] R. Achanta, A. Shaji, K. Smith, A. Lucchi, P. Fua, and Sabine Süsstrunk. SLIC superpixels compared to state-of-the-art superpixel methods. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, page 2274–2282, 2012. 3

[8] Marco Cuturi and Mathieu Blondel. Soft-DTW: a differentiable loss function for time-series. In *International conference on machine learning*, pages 894–903. PMLR, 2017. 2

[9] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16×16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020. 3

[10] Tzu-Mao Li, Michal Lukáč, Michaël Gharbi, and Jonathan Ragan-Kelley. Differentiable vector graphics rasterization for editing and learning. *ACM Transactions on Graphics (TOG)*, 39(6):1–15, 2020. 4

[11] Zhengqin Li and Jiansheng Chen. Superpixel segmentation using linear spectral clustering. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1356–1363, 2015. 3

[12] Xu Ma, Yuqian Zhou, Xingqian Xu, Bin Sun, Valerii Filev, Nikita Orlov, Yun Fu, and Humphrey Shi. Towards layer-wise image vectorization. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 16314–16323, 2022. 4

[13] Pradyumna Reddy, Michael Gharbi, Michal Lukac, and Niloy J Mitra. Im2Vec: Synthesizing vector graphics without vector supervision. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7342–7351, 2021. 3

[14] Peter Selinger. Potrace: a polygon-based tracing algorithm, 2003. 4

[15] Michael Van den Bergh, Xavier Boix, Gemma Roig, Benjamin De Capitani, and Luc Van Gool. SEEDS: Superpixels extracted via energy-driven sampling. In *Computer Vision–ECCV 2012: 12th European Conference on Computer Vision (ECCV)*, pages 13–26. Springer, 2012. 3

[16] Zhou Wang, Alan C Bovik, Hamid R Sheikh, and Eero P Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE Transactions on Image Processing*, 13(4):600–612, 2004. 3

[17] Fengting Yang, Qian Sun, Hailin Jin, and Zihan Zhou. Superpixel segmentation with fully convolutional networks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 13964–13973, 2020. 3

[18] Richard Zhang, Phillip Isola, Alexei A Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 586–595, 2018. 3

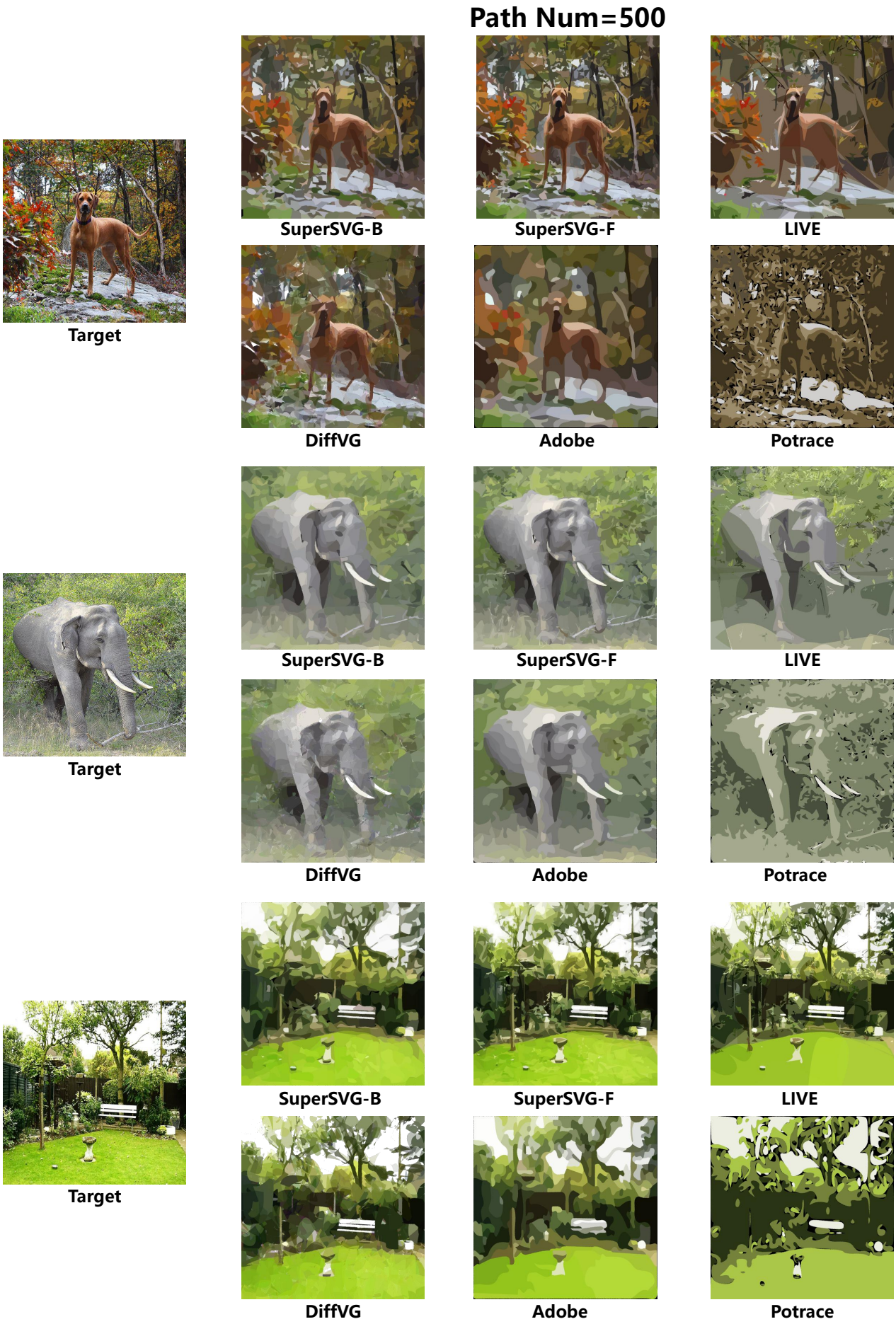

Figure 5. More comparison with the state-of-the-art methods under 500 SVG paths.

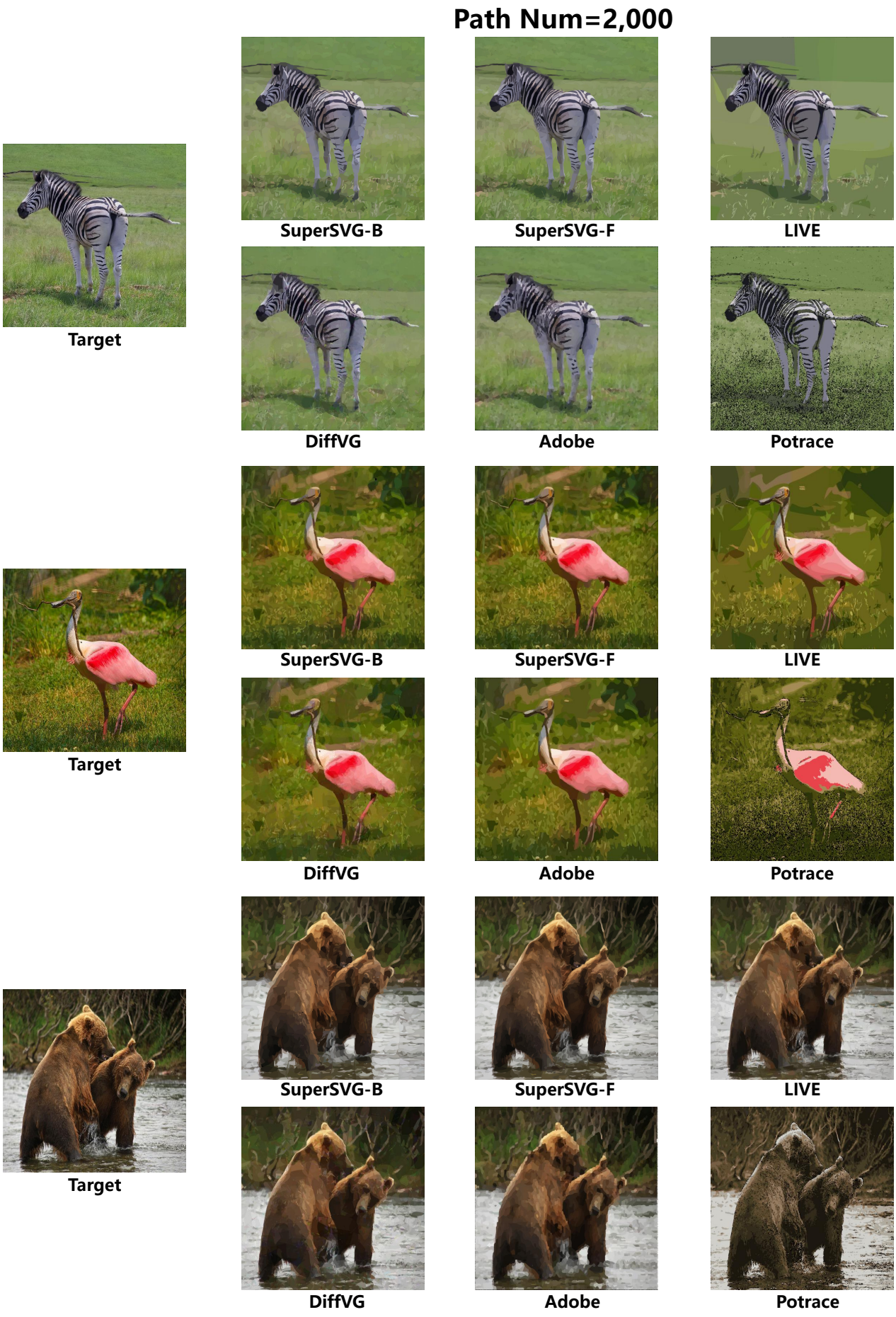

Figure 6. More comparison with the state-of-the-art methods under 2,000 SVG paths.

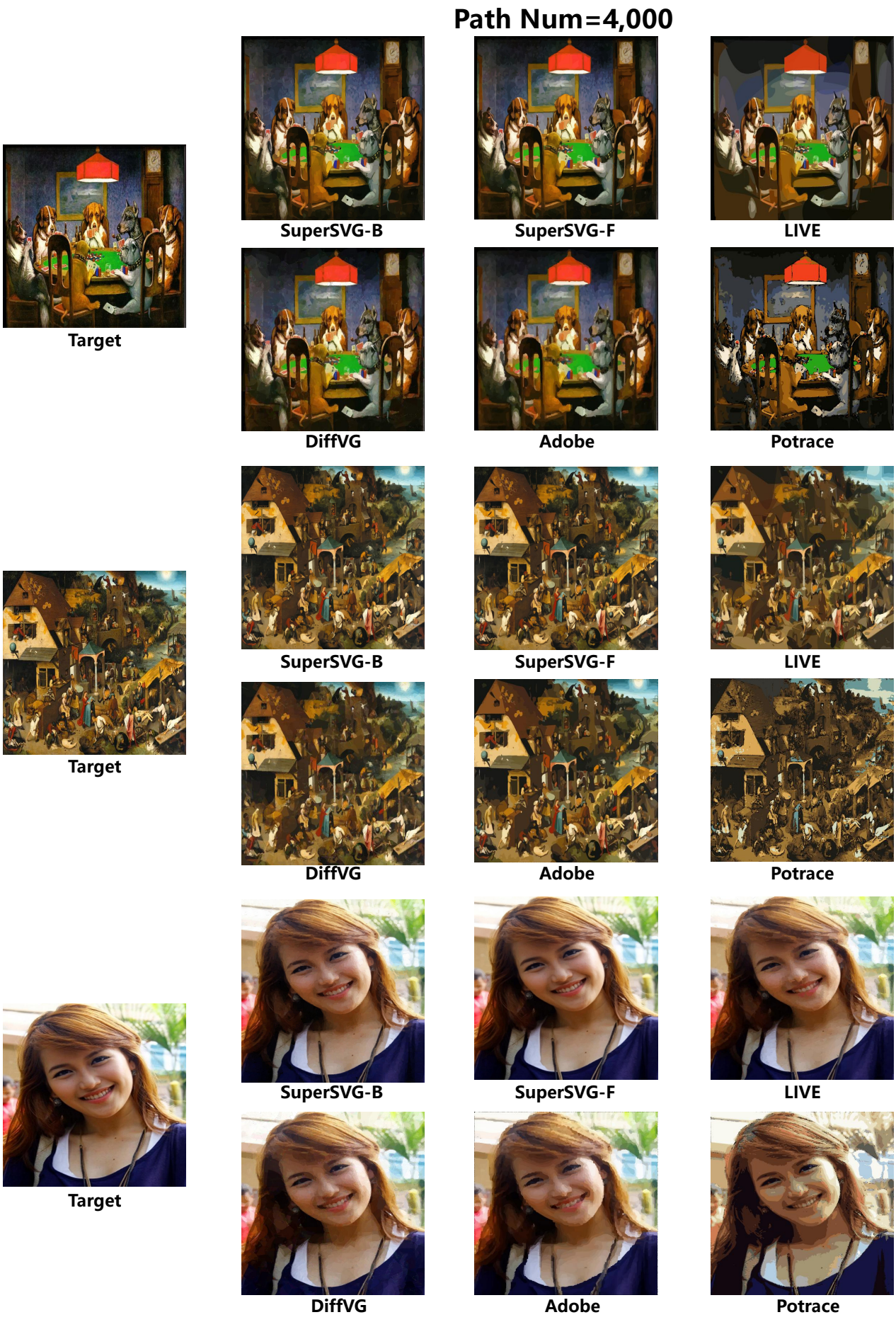

Figure 7. More comparison with the state-of-the-art methods under 4,000 SVG paths.

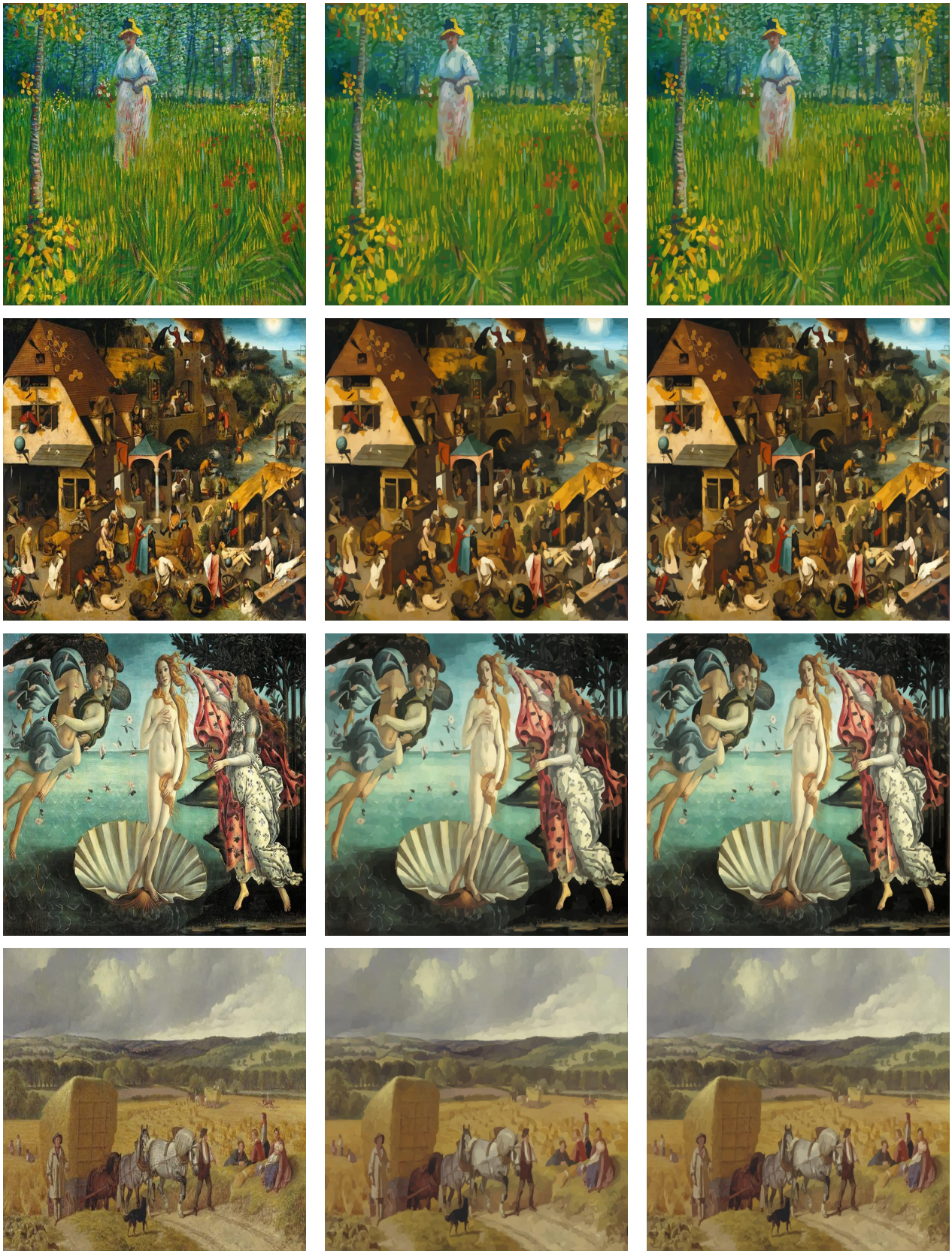

Figure 8. More of our results under 4,000 paths.

Figure 9. More of our results under 4,000 paths.

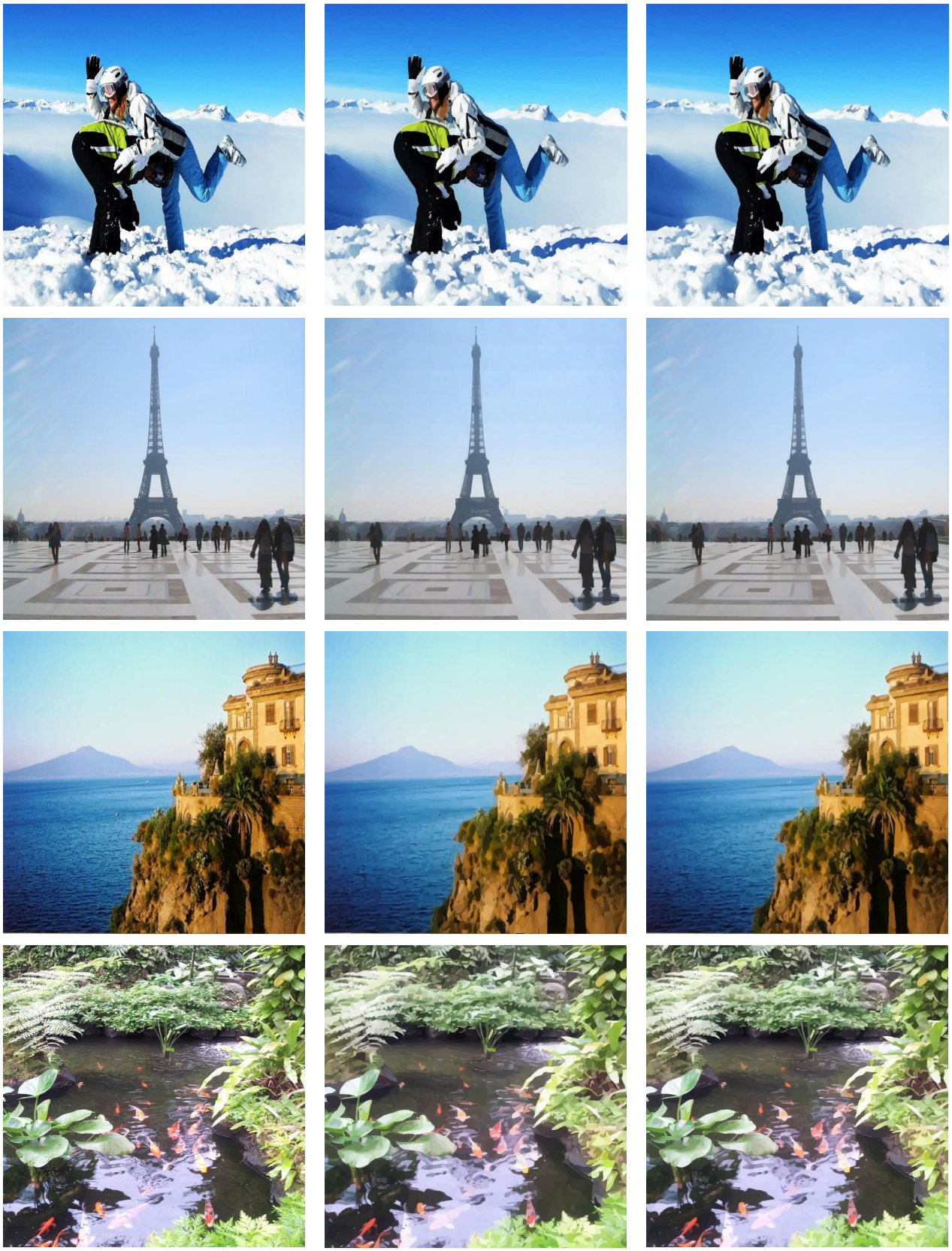

Figure 10. More of our results under 4,000 paths.

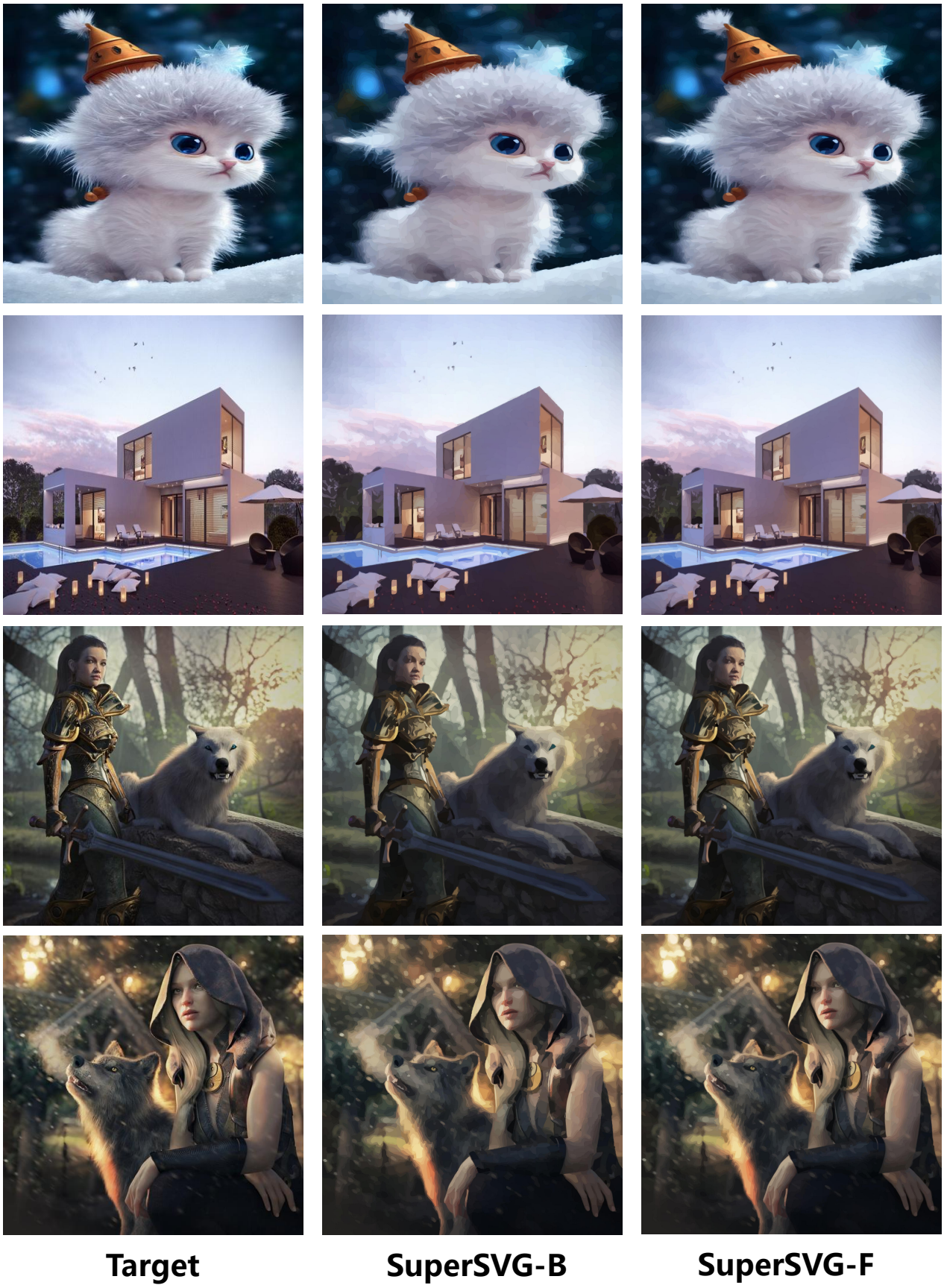

Figure 11. More of our results under 4,000 paths.