SVGDreamer: Text Guided SVG Generation with Diffusion Model

Supplementary Material

Overview

This supplementary material is organized into several sections that provide additional details and analysis related to our work on SVGDreamer. Specifically, it will cover the following aspects:

- In section A, we present additional qualitative results of SVGDreamer, demonstrating its ability to generate SVGs with high editability, visual quality, and diversity.
- In section B, we demonstrate the potential applications of SVGDreamer in poster design and icon design.
- In section C, we provide more implementation details of SVGDreamer.
- In section D, We explain how to identify semantic objects in SIVE prompts.
- In section E, we conduct additional ablation studies to demonstrate the effects of CFG weights (see Appendix E.1), ReFL (see Appendix E.2), the number of vector particles (see Appendix E.3), and the number of paths (see Appendix E.4).
- In section F, we provide example results from using VPSD for raster image synthesis.
- In section G, we show the pseudo code of SVGDreamer. Code is available now [1].

A. Additional Qualitative Results

Editability. Our tool, SVGDreamer, is designed to generate high-quality vector graphics with versatile editable properties, empowering users to efficiently reuse synthesized vector elements and create new vector graphics. In our manuscript, Fig. 5 showcases two posters where each character is generated using SVGDreamer. Additionally, we present further examples in Fig. 7. These generated SVGs can be decomposed into background and foreground elements, which can then be recombined to create new SVGs.

Visual Quality and Diversity. In Fig. 8, we present additional examples generated by SVGDreamer, showcasing its ability to synthesize diverse object-level and scene-level vector graphics based on text prompts. Notably, our model can generate vector graphics with different styles, such as oil painting, watercolor, and sketch, by manipulating the type of primitives and text prompts. By incorporating the VPSD and ReFL into our model, SVGDreamer produces richer details compared to the state-of-the-art method VectorFusion.

It is important to highlight that our model can achieve different styles without relying on additional reference style images. Existing approaches for generating stylized vector graphics, such as StyleClipDraw, typically follow a style transfer pipeline used for raster images, which requires an additional style image as a reference. In contrast, SVGDreamer, being built upon a T2I model, can simply inject style information through text prompts. For instance, in the second example, we can obtain an oil painting in Van Gogh's style by using a text prompt.

B. Applications of SVGDreamer

In this section, we will demonstrate the utilization of SVGDreamer for synthesizing vector posters and icons. **Poster Design.** A poster is a large sheet used for advertising events, films, or conveying messages to people. It usually contains text and graphic elements. While existing T2I models have been developing rapidly, they still face challenges in text generation and control. On the other hand, SVG offers greater ease in text control. In Fig. 9, we compare the posters generated by our SVGDreamer with those produced by four T2I models. It is important to note that all results generated by these T2I models are in raster format.

We will start by explaining the usage of our SVGDreamer tool for poster design. Initially, we employ SVGDreamer to generate graphic content. Then, we utilize modern font libraries to create vector fonts, taking advantage of SVG's transform properties to precisely control the font layout. Ultimately, we combine the vector images and fonts to produce comprehensive vector posters. To be more specific, we employ the FreeType font library [2] to represent glyphs using vectorized graphic outlines. In simpler terms, these glyph's outlines are composed of lines, Bézier curves, or B-Spline curves. This approach allows us to adjust and render the letters at any size, similar to other vector illustrations. The joint optimization of text and graphic content for enhanced visual quality is left for future work.

As depicted in Fig. 9, both Stable Diffusion [26] (the first column) and DeepFloyd IF [37] (the second column) display various text rendering errors, including missing glyphs, repeated or merged glyphs, and misshapen glyphs. GlyphControl [50] (the third column) occasionally omits individual letters, and the fonts obscure content, resulting in areas where the fonts appear to lack content objects. TextDiffuser [2] (the fifth column) is capable of generating fonts for different layouts, but it also suffers from the artifact of layout control masks, which disrupts the overall harmony of the content. In contrast, posters created using our SVGDreamer are not restricted by resolution size, ensuring the

[1] https://github.com/ximinng/SVGDreamer

[2] http://freetype.org/index.html

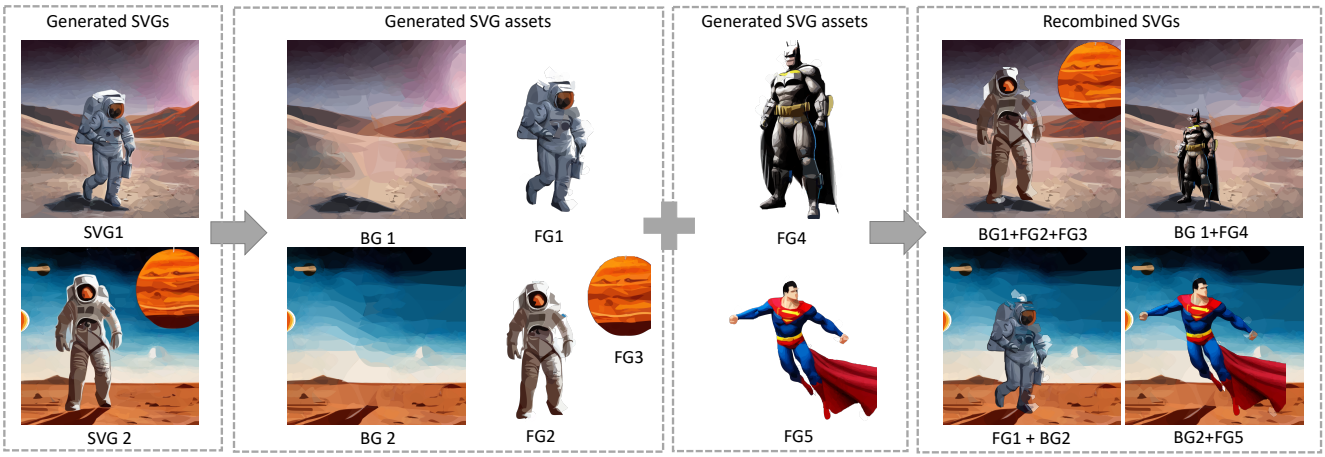

Figure 7. Examples showcasing the editability of the results generated by our SVGDreamer.

text remains clear and legible. Moreover, our approach offers the convenience of easily editing both fonts and layout, providing a more flexible poster design approach.

Icon Design. In addition to posters, SVGDreamer can be applied in icon design (as shown in the Fig. 10). We use SVGDreamer to obtain the graphic contents, and then create the polygon and circle layout by defining *def* tags in the SVG file. Then, we append the vector text paths to the end of the SVG file in order to obtain a complete vector icon.

C. Implementation Details

Our method is based on the pre-trained Stable Diffusion model [26]. We use the Adam optimizer with $\beta_1 = 0.9$, $\beta_2 = 0.9$, $\epsilon = 1e-6$ for optimizing SVG path parameters $\theta = \{P_i, C_i\}_{i=1}^{n}$. We use a learning rate warm-up strategy. In the first 50 iterations, we gradually increase the control point learning rate from 0.01 to 0.9, and then employ exponential decay from 0.8 to 0.4 in the remaining 650 iterations (a total of 700 iterations). For the color learning rate, we set it to 0.1 and the stroke width learning rate to 0.01. We adopt AdamW optimizer with $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 1e-10$, $lr = 1e-5$ for the training of LoRA [10] parameters. In the majority of our experiments, we set the particle number k to 6, which means that 6 particles participate in the VPSD (Sec. 3.2), LoRA update, and ReFL update simultaneously. To ensure diversity and fidelity to text prompts in the synthesized SVGs, while maintaining rich details, we set the guidance scale of the Classifier-free Guidance (CFG [7]) to 7.5. During the optimization process, SVGDreamer requires at least 31 GB memory on an Nvidia-V100 GPU to produce 6 SVGs.

Synthesizing flat iconographic vectors, we allow path control points and fill colors to be optimized. During the course of optimization, many paths learn low opacity or shrink to a small area and are unused. To encourage usage of paths and therefore more diverse and detailed images, motivated by VectorFusion [12], we periodically reinitialize paths with fill-color opacity or area below a threshold. Reinitialized paths are removed from optimization and the SVG, and recreated as a randomly located and colored circle on top of existing paths.

D. Object Identification in SIVE Prompts

It is common for multiple nouns within a sentence to refer to the same object. We present two examples in Fig. 11. In our experiments, we did not employ a specific selection strategy because the cross-attention maps for such nouns-for example, "man" and "astronaut" – are very similar. Therefore, choosing either "man" or "astronaut" produces similar results with our method. For more precise control, users may utilize the cross-attention maps of the text prompt to identify the desired objects. In SIVE, users can use visual text prompts to identify semantic objects.

E. Additional Ablation Studies

Next, we provide additional ablation experiments to demonstrate the effectiveness of the proposed components.

E.1. Ablation on CFG [7] Weights

In this section, we explore how Classifier-free Guidances (CFG) [7] affects the diversity of generated results. For VPSD, we set the number of particles as 6 and run experiments with different CFG values. For LSDS [12], we run 4 times of generation with different random seeds. The results are shown in Fig. 12. As shown in the figure, smaller CFG provides more diversity. We conjecture that this is because the distribution of smaller guidance weights has more diverse modes. However, when the CFG becomes too small (e.g., CFG= 2), it cannot provide enough guidance to generate reasonable results. Therefore, in our implementation, we set CFG to 7.5 as a trade-off between diversity and optimization stability. Note that SDS-based methods [12, 48] do not work well in such small CFG weights. Instead, our

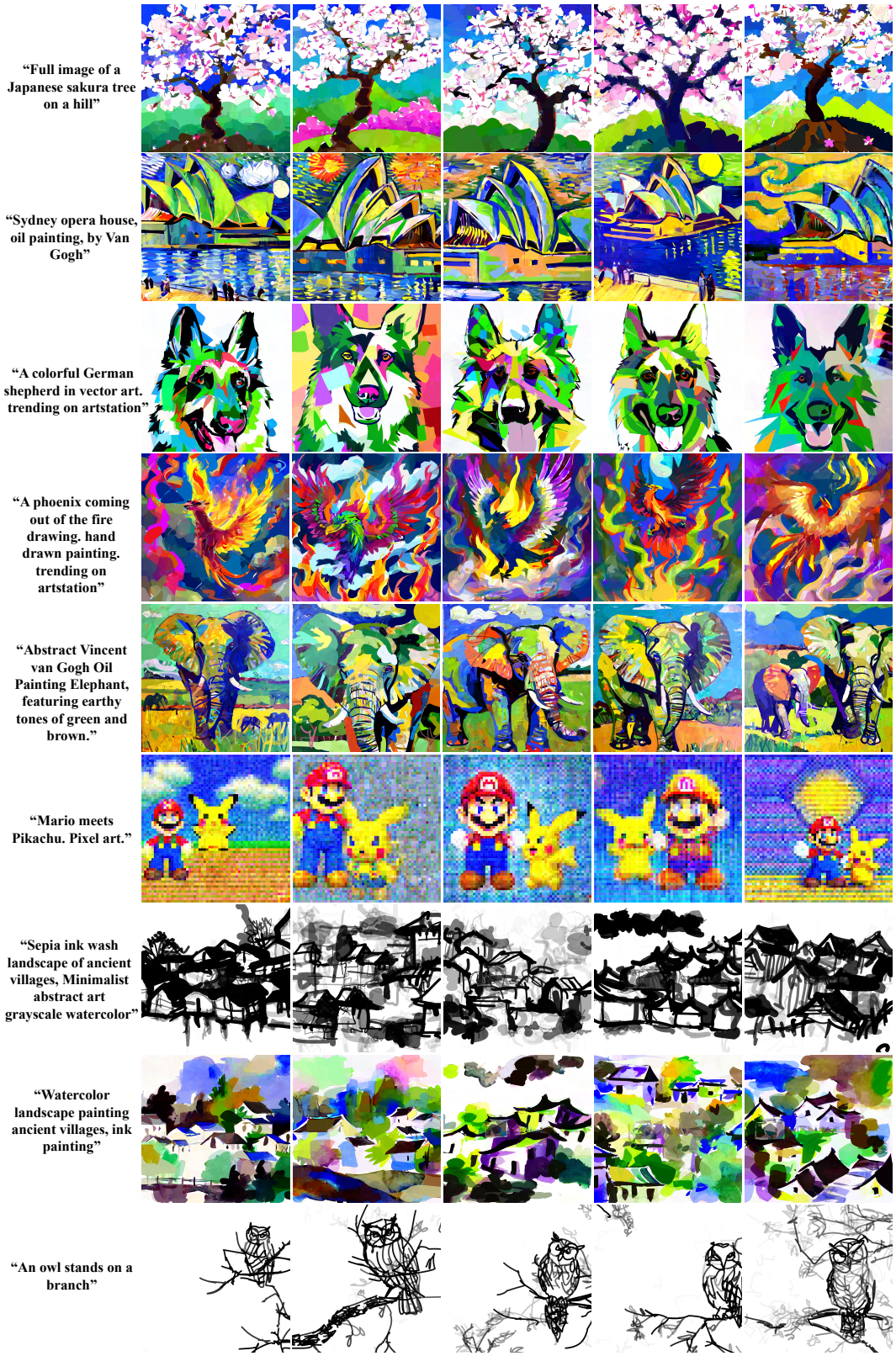

Figure 8. More results generated by our SVGDreamer.

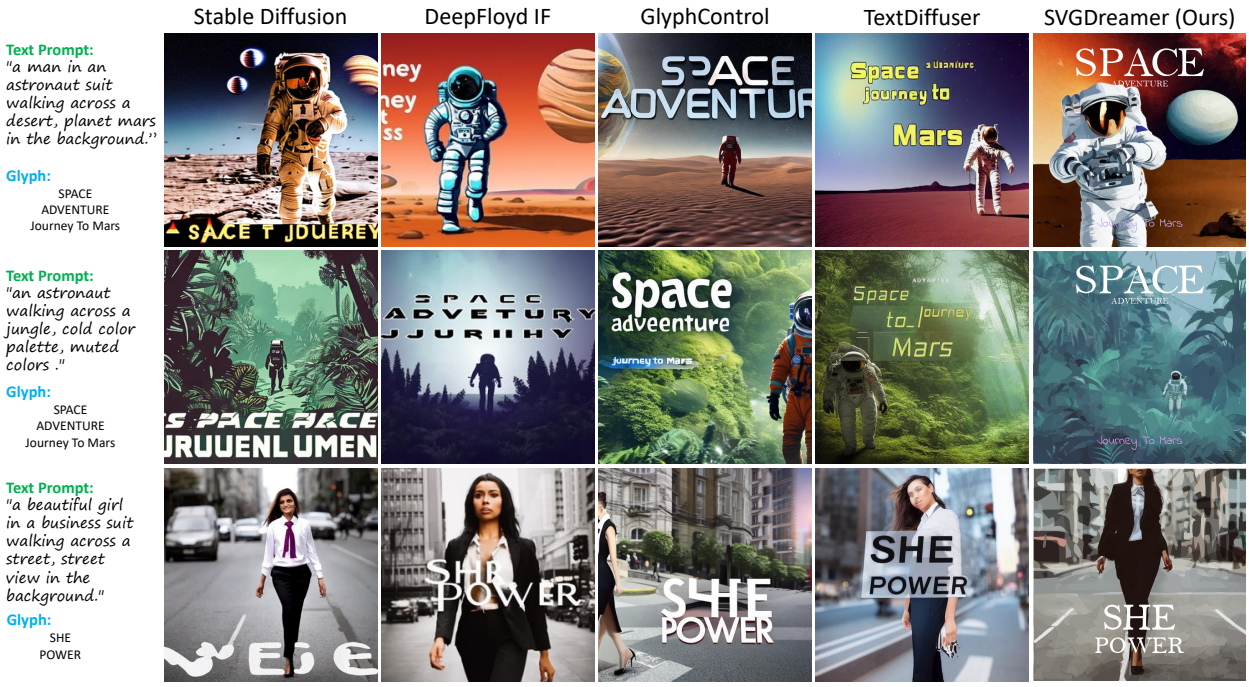

Figure 9. Comparison of synthetic posters generated by different methods. The input text prompts and glyphs to be added to the posters are displayed on the left side.

Figure 10. Examples of synthetic icons. Note that the glyphs are manually added.

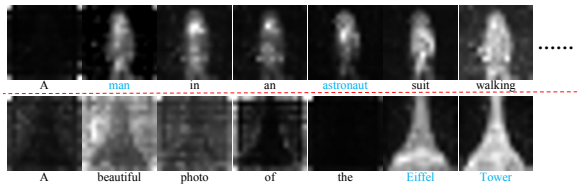

Figure 11. Visualizations of the cross-attention maps.

VPSD provides a trade-off option between CFG weight and diversity, and it can generate more diverse results by simply setting a smaller CFG.

E.2. Ablation on ReFL

In [45], only selected particles update the LoRA network in each iteration. However, this approach neglects the learning progression of LoRA networks, which are used to represent variational distributions. These networks typically require numerous iterations to approximate the optimal distribution, resulting in slow convergence. Unfortunately, the randomness introduced by particle initialization can lead to early learning of sub-optimal particles, which adversely affects the final convergence result. In VPSD, we introduce a Reward Feedback Learning (ReFL) method. This method leverages a pre-trained reward model [49] to assign reward scores to samples collected from LoRA model. Then LoRA model subsequently updates from these reweighted samples. As indicated in Table 2, this led to a significant reduction in the number of iterations by almost 50%, resulting in a 50% decrease in optimization time. And improves the aesthetic score of the SVG by filtering out samples with low reward values in LoRA. Filtering out samples with low reward values, as demonstrated in Table 1, enhances the aesthetic score of the SVG. The visual improvements brought by ReFL are illustrated in Fig. 13.

E.3. Ablation on the Number of Vector Particles

we investigate the impact of the number of particles on the generated results. We vary the number of particles in 1, 4, 8, 16 and analyze how this variation affects the outcomes. The CFG of VPSD is set as 7.5. As shown in Fig. 14, the di-

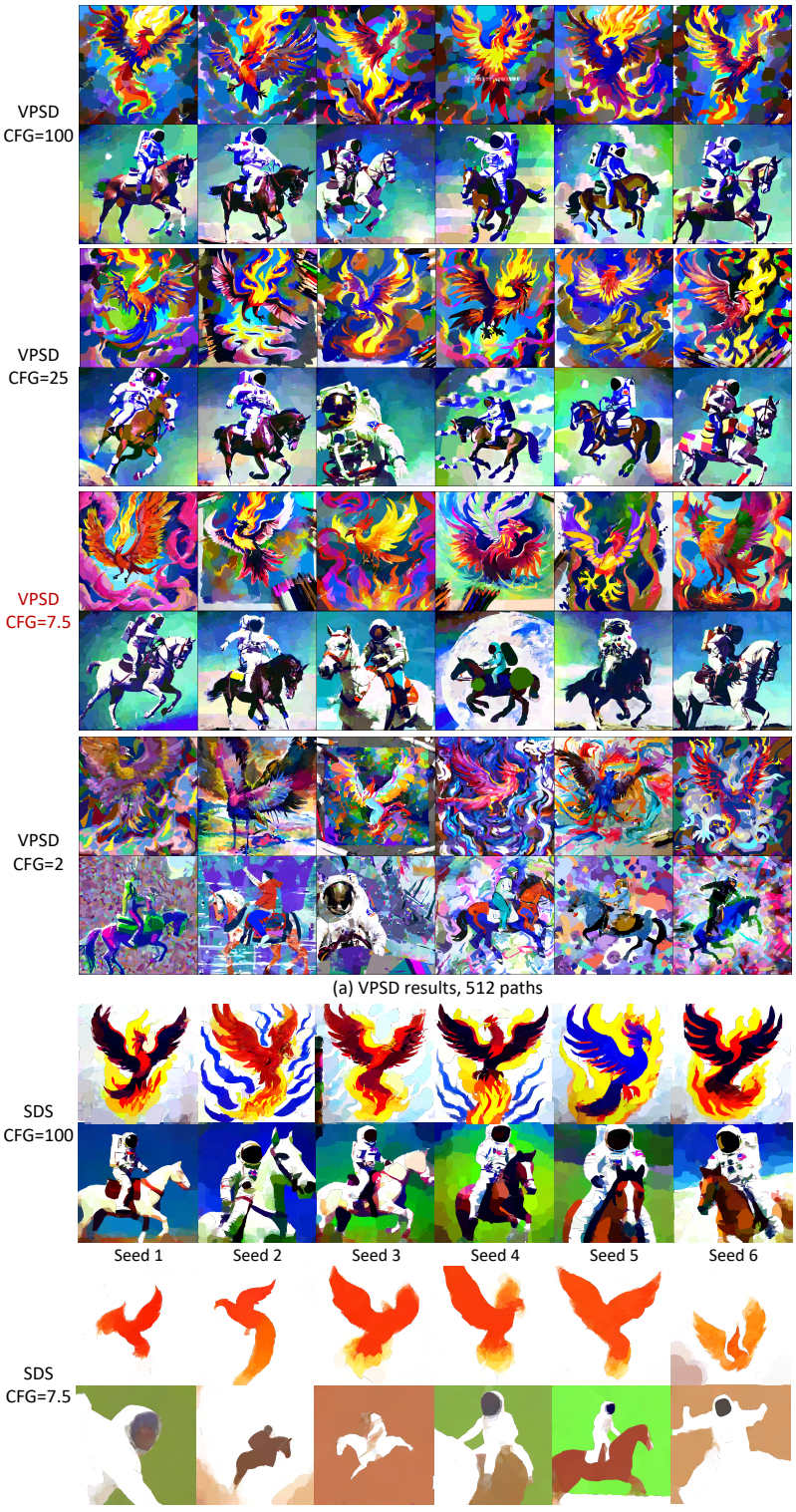

Figure 12. Ablation on how Classifier-free Guidances (CFG) [7] weight affects the randomness. Smaller CFG provides more diversity. But too small CFG provides less optimization stability. The prompt is "A photograph of an astronaut riding a horse".

versity of the generated results is slightly larger as the number of particles increases. Meanwhile, the quality of generated results is not significantly affected by the number of particles. Considering the high computation overhead asso-

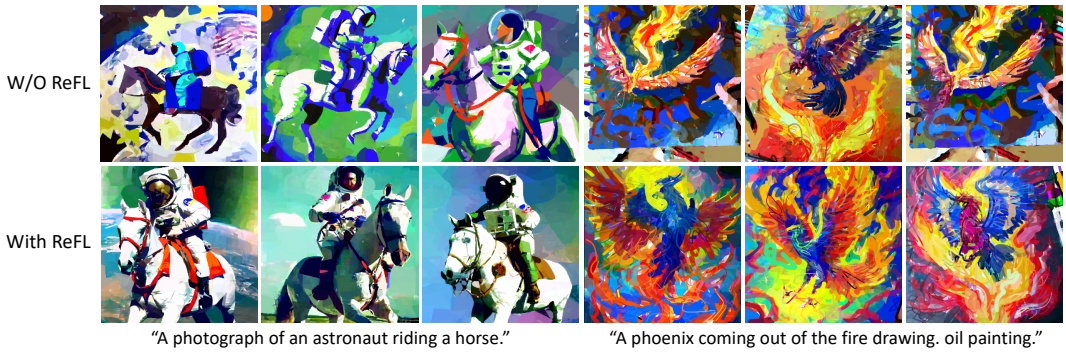

Figure 13. Effect of the Reward Feedback Learning (ReFL) on the generated results. When employing ReFL, the visual quality of the generated results is significantly enhanced.

Figure 14. Ablation on the number of particles. The diversity of the generated results is slightly larger as the number of particles increases. The quality of generated results is not significantly affected by the number of particles. The prompt is "A photograph of an astronaut riding a horse".

Table 2. Efficiency of our proposed ReFL in SVGDreamer.

Method	Canvas Size	Path Number	Iteration Steps	Time(min:sec)
W/O ReFL	224 * 224	128	500	13m15s
W ReFL	224 * 224	128	300	6m45s
W/O ReFL	600 * 600	256	500	14m21s
W ReFL	600 * 600	256	300	7m21s

ciated with optimizing vector primitive representations and the limitations imposed by available computation resources, we limit our testing to a maximum of 6 particles.

E.4. Ablation on the Number of Paths

This subsection analyzes the effect of different stroke numbers on VPSD synthetic vector images. Figure 15 shows examples with 128, 256, 512, and 768 paths, from top to bottom, using Iconography primitives. As the path count increases, the image transitions from abstract to more concrete, and the level of detail notably improves. VPSD offers

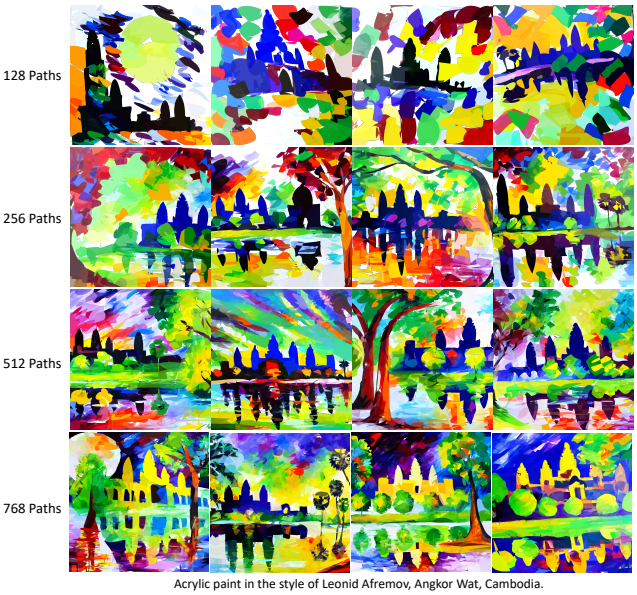

Figure 15. Effect of the number of paths on synthesized results.

Figure 16. Comparison of the results from using VPSD and VSD for 2D image synthesis.

superior visual details compared to SDS, including aspects like water reflections. Additionally, VPSD better aligns with text prompts.

F. VPSD for 2D Image Synthesis

In this work, VPSD is specifically designed for text-to-SVG generation; however, it can also be adapted for 2D image synthesis. As illustrated in Fig. 16, images synthesized by VSD may exhibit displaced or incomplete object layouts, resulting in samples that might not meet human aesthetic preferences. In contrast, VPSD integrates a reward score within its feedback learning process, which significantly enhances the quality of the generated images.

G. Algorithm for VPSD

We summarize the algorithm of Vectorized Particle-based Score Distillation (VPSD) in Algorithm 1. First, VPSD initializes $k(\geq 1)$ groups of SVG parameters, a pretrained diffusion model ϵ_ϕ parameterized by ϕ and the LoRA layers $\epsilon_{\phi_{\text{est}}}$ parameterized by ϕ_{est}, as the pretrained reward model r. Note that only the diffusion model is pretrained with frozen parameters, while LoRA [10] thaws some of its parameters. Subsequently, VPSD randomly selects a parameter θ from the set of SVG parameters and generates a raster image x based on this selection. The parameter θ is then updated using Variational Score Distillation (VSD). k samples are sampled using $\epsilon_\phi(y)$ and utilized to update the parameters of ϕ. This process is repeated until a satisfactory result is obtained and the algorithm returns k groups of SVG parameters as the final output.

Algorithm 2 is the combination of VPSD and SIVE (Semantic-driven Image Vectorizatio). This algorithm has the same initialization as VPSD, but it needs to get a sample using diffusion model ϵ_ϕ given text prompt y. In the sampling process, it can obtain the sample's corresponding attention map. Depending on attention map, the algorithm can get background mask and foreground mask. It optimizes the SVG parameters according to the foreground mask and background mask, respectively, and then finetunes them using the VPSD algorithm.

Algorithm 1 Vectorized Particle-based Score Distillation (VPSD)

Require: Text prompt y. Number of particles k (≥ 1). Number of SVG primitives n (≥ 1). Pretrained Text-to-Img Diffusion Model ϵ_ϕ. Learning rates η_p for SVG path parameters. Learning rate η_e for diffusion model parameters. r represents the pretrained reward model [49]. λ_r indicates reward feedback strength.

Initialize: k groups of SVG parameters $\{\theta^{(1)}, \cdots, \theta^{(n)}\} = \{(P_j^{(i)}, C_j^{(i)})\}_{j=1}^n$, a pretrained diffusion model ϵ_ϕ is parameterized by ϕ, a LoRA [10] model $\epsilon_{\phi_{\text{est}}}$ is parameterized by ϕ_{est}, the pretrained reward model r.
while not converged **do**
  Randomly sample $\theta \sim \{\theta^{(i)}\}_{i=1}^k$.
  Render the SVG parameter θ to get a raster image $x = \mathcal{R}(\theta)$.
  $\theta \leftarrow \theta - \eta_p \mathbb{E}_{t,\epsilon,p,c}\left[\omega(t)(\epsilon_\phi(\mathbf{z}_t; y, t) - \epsilon_{\phi_{\text{est}}}(\mathbf{z}_t); y, p, c, t)\frac{\partial \mathbf{z}}{\partial \theta}\right]$
  Sample $w(\leq k)$ samples using $\epsilon_{\phi_{\text{est}}}(y)$.
  $\phi \leftarrow \phi - \eta_e \nabla_\phi \left[\mathbb{E}_{\epsilon,t}\left\|\epsilon_{\phi_{\text{est}}}(\mathbf{z}_t; y, p, c, t) - \epsilon\right\|_2^2 + \lambda_r \mathbb{E}_{y,w}\left[\psi(r(y, g_{\phi_{\text{est}}}(y)))\right]\right]$
end while
return $\{\theta_1, \cdots \theta_k\}$.

Algorithm 2 Semantic-driven Image Vectorization (SIVE) + VPSD

Require: Text prompt y. Number of particles k (≥ 1). Number of SVG primitives n (≥ 1). Pretrained Text-to-Img Diffusion Model ϵ_ϕ. Learning rates η_p for SVG path parameters. Learning rate η_e for diffusion model parameters. r represents the pretrained reward model [49]. λ_r indicates reward feedback strength.

Initialize: k groups of SVG parameters $\{\theta^{(1)}, \cdots, \theta^{(n)}\} = \{(P_j^{(i)}, C_j^{(i)})\}_{j=1}^n$, a noise prediction model ϵ_ϕ parameterized by ϕ.
Sample a sample using $\epsilon_\phi(y)$.
Get the attention map corresponding to the i-th text token $\mathcal{M}_{\text{FG}}^i = \text{softmax}(QK_i^T)/\sqrt{d}$
Get the background attention map $\mathcal{M}_{\text{BG}} = 1 - (\sum_{i=1}^O \mathcal{M}_{\text{FG}}^i)$
Get the background mask and foreground masks $\hat{\mathcal{M}} = \{\{\hat{\mathcal{M}}_{\text{FG}}\}_{o=1}^O, \hat{\mathcal{M}}_{\text{BG}}\}$, respectively.
while not converged **do**
  $\theta^{(1)} \leftarrow \theta^{(1)} - \eta_p \nabla_\theta \mathbb{E}_o(\hat{\mathcal{M}}_i \odot I - \hat{\mathcal{M}}_i \odot \mathbf{x})^2$
end while
Initialize: a LoRA [10] model $\epsilon_{\phi_{\text{est}}}$ is parameterized by ϕ_{est}, the pretrained reward model r.
while not converged **do**
  Randomly sample $\theta \sim \{\theta\}_{i=1}^k$.
  Render the SVG parameter θ to get a raster image $x = \mathcal{R}(\theta)$.
  $\theta \leftarrow \theta - \eta_p \mathbb{E}_{t,\epsilon,p,c}\left[\omega(t)(\epsilon_\phi(\mathbf{z}_t; y, t) - \epsilon_{\phi_{\text{est}}}(\mathbf{z}_t); y, p, c, t)\frac{\partial \mathbf{z}}{\partial \theta}\right]$
  Sample $w(\leq k)$ samples using $\epsilon_{\phi_{\text{est}}}(y)$.
  $\phi \leftarrow \phi - \eta_e \nabla_\phi \left[\mathbb{E}_{\epsilon,t}\left\|\epsilon_{\phi_{\text{est}}}(\mathbf{z}_t; y, p, c, t) - \epsilon\right\|_2^2 + \lambda_r \mathbb{E}_{y,w}\left[\psi(r(y, g_{\phi_{\text{est}}}(y)))\right]\right]$
end while
return $\{\theta_1, \cdots \theta_k\}$.