CraftSVG: Multi-Object Text-to-SVG Synthesis via Layout Guided Diffusion

Supplementary Material

6. Implementation details

We implemented our model using PyTorch [52] and leveraged the differentiable rasterizer framework, diffvg[1]. We utilize the diffusion U-Net of the GLIGEN [53] model without further training or adaptation. This selected GLIGEN model used stable diffusion v1.4 from diffusers[2] as a baseline with additional layout guidance. We set this guidance scale to 7.5, as mentioned in [53]. For the energy minimization in image synthesis, we repeat the optimization step 5 times for every object denoising step until the repetition is reduced to 1. Also, the k in Topk$_u$ in Eq. (4) is set to 203% of the area of the mask for each mask. The background part (second term) of Eq. (4) is

For MLP-based abstraction, we set the number of strokes $n <= 64$ in the first phase of object sketching via MLP_s and train it for 500 iterations. Further, to perform the background abstraction for generating a simplified sketch, we perform 100 iterative steps of MLP_d. Then along with $\mathcal{L}_{align}$ we define a separate step size for sampling the features of $S_{combined}$ For simplifying the $S_{combined}$, we set this step size to be 0.3, 0.4, 0.6, 0.7 for layers 2, 7, 8, 11, respectively for both MLP_s and MLP_d. Each simplification step is obtained through optimizing the $\mathcal{L}_{synth}$ and backpropagating the gradient descent to both MLP Networks via differential rendering $\mathcal{R}_d$. Here, we employ the Adam optimizer with a constant learning rate of 1e-5. (NOTE: Both the MLP networks have 11 layers, and the first 5 layers consist of 512 neurons. We gradually decrease the width of the network such that the 6th and 7th layers have only 256 neurons. Similarly, the 8th, 9th, and 10th layers have only 128 neurons, and the final layers have only 64 neurons.

Now, in order to optimize the SVG parameters θ, we extract the vector features from the rasterized image I_r and differential renderer $\mathcal{R}_d(\theta)$ via an image encoder $\mathcal{I}$. We use the CLIP-VIT B-32 [54], which provides global self-attention and cross-attention along with positional embeddings. We set the iterations to 500 to optimize $\mathcal{L}_{\text{synth}}$ to obtain the final SVG. Now, in terms of initialization, we follow the same initialization strategy as of [35] in order to initialize the primitive shapes with semantic control (*i.e.*, instead of dividing the canvas into patches, we only initialize the shapes in the foreground region obtained by per-box mask latent-based attention mechanism). For the Bézier curve, the initialization is even simpler. We just decide the number of keypoints in the foreground region and initialize a single curve from each of the keypoints with a varying number of control points (we keep it random to provide more flexibility). Also, it has to be noted that, we didn't perform any augmentations mentioned in [1, 35] as the CLIP ViT B-32 [54] model takes care of perspective distortions which also helps us to provide faster convergence than [1, 3, 4, 26, 35]. All the experiments has been performed in a single NVIDIA A6000 Ada generation 48GB GPU. (Note: as CraftSVG is a training-free approach, we haven't used any large-scale dataset like StarVector [55]. Also, as our main focus is multi-object SVG generation, we exclude the training-based SVG icon generation approaches [56, 57])

[1] https://github.com/BachiLi/diffvg

[2] https://github.com/huggingface/diffusers

7. Layout generation and correction

In every NLP task, the main challenge is to decide the prompt/ instructions we should provide the language model to solve the task. For layout generation, we divide this prompt into five sections namely, (1) Task Specification, (2) Supporting details, (3) In-context learning, (4) Layout generation, and (5) Layout correction. The details of each section are specified as follows:

Task specification: This section describes the prompts for explaining the layout generation task to the LLM. To do that, we use the following prompt. This part is quite similar to the instructions mentioned in [27].

> You are an intelligent bounding box generator. I will provide you with a caption for a photo, image, or painting. Your task is to generate the bounding boxes for the objects mentioned in the caption, along with a background prompt describing the scene.

Instruction details: This section mentions the details about canvas size, desired layout format, the errors we have used to optimize the layout, and so on.

> The canvases are of size 512x512. The top-left corner has coordinates [0, 0]. The bottom-right corner has coordinates [512, 512]. The bounding boxes should not overlap or go beyond the image boundaries. Each bounding box should be in the format of (object name, [top-left x coordinate, top-left y coordinate, box width, box height]) and include exactly one object (i.e., start the object name with "a" or "an" if possible). Do not put objects already provided in the bounding boxes into the background prompt. Do not include non-existing or excluded objects in the background prompt. If needed, you can make reasonable guesses. Your main goal is to minimize the error between the user-provided prompt and the reconstructed descriptions by maximizing the cosine and Jaccard similarity and minimizing the Levenshtein distance. Initially, the error is one. Your goal is to reduce the error close to zero. Please refer to the example below for the desired format.

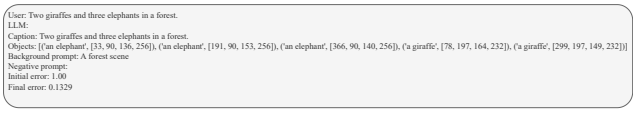
We have used some additional prompts over [27] (highlighted in blue color), which help us to enable the layout correction at a further stage.

In-context learning: Here, we provide some basic templates, which LLM should follow during layout generation. Those examples are acquired from the MS-COCO dataset [58] with human specification (*i.e.* in MS-COCO they only specified object name, here we also specified some characteristics of the object) to clarify the layout representation and obtained preferences to disperse that ambiguity. Some example templates are provided as follows:

On the top of [27], we have mentioned our desired final error score. Based on this template, LLM generates its initial layout based on the given captions. It has to be noted

that during this in-context learning, we ensured that each object instance is represented by a single bounding box, and no foreground objects are specified in the boxes to the background caption to ensure all foreground objects contribute to the per-box mask latent-based canvas initialization.

Layout generation: This step provides interactions between users and the LLM. Here, users provide a caption, and the LLM generates its corresponding bounding boxes. An example has been provided as follows:



Layout correction: In the last step, we have performed iterative layout correction through the caption reconstruction error proposed in Eq. (1). The detailed algorithm has been depicted in Algorithm 1.

8. Transformations affect shape deformation

As we have discussed earlier, the Bézier curves are transformed by projective transformations which have 8 degrees of freedom (dof) with greater flexibility, and the primitive shapes are transformed via affine transformation which has 6 dof and more control points than the Bézier curves which jointly restrict their transformation flexibility. This experiment aims to explain the effect of projective transformations on primitive shapes and why we cannot use them during canvas completion with primitive shapes. Fig. 14 shows the effect of affine as well as projective transformations on a simple trapezoidal like shape made of simple primitives.

It has been observed that, after an affine transformation, shapes are only rotated, scaled, or translated however, they maintain their shape characteristics. This is because an affine transformation is defined by a 3×3 matrix of

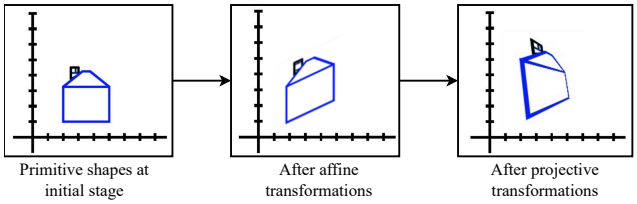


Fig. 14. **Affine and projective transformations on primitive shapes:** All the primitive shapes get deformed in projective transformations via morphing and gradually lose their recognizability.

the following form: $\text{A} = \begin{bmatrix} A_{11} & A_{12} & t_x \\ A_{21} & A_{22} & t_y \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} A & t \\ 0^T & 1 \end{bmatrix}$, where $t = (t_x, t_y)$ is the translation vector comprising the amount of translation along the x-axis and y-axis, respectively. Now, in order to understand the transformations provided by the matrix 2×2 square matrix A, we perform a singular value decomposition (SVD) of A as follows:

$$A \xrightarrow{\text{SVD}} USV^T = (UV^T)(VSV^T) = R(\theta)R(-\phi)SR(\phi) \quad (9)$$

where $UV^T = R(\theta)$ and $V^T = R(\phi)$. This SVD always provides two orthonormal matrices U and V and a diagonal matrix S containing the eigenvalues of the matrix A. As the rotation matrix is also orthonormal, these two orthogonal matrices depict two rotations and provide 2 degrees of freedom. on the other hand, S is a diagonal matrix containing 2 eigenvalues of A (As A is a 2×2 matrix); each eigenvalue represents one scaling factor providing 1+1 = 2 degrees of freedom. As we do not get any morphing operation via SVD decomposition, the characteristics of the shape will be preserved after deformation, which helps the player to identify it in the final canvas.

Similarly, projective transformations are also performed with a 3×3 matrix of the form $P = \begin{bmatrix} A_{11} & A_{12} & t_x \\ A_{21} & A_{22} & t_y \\ v_{11} & v_{12} & v_{13} \end{bmatrix} = \begin{bmatrix} A & t \\ v^T & v \end{bmatrix}$, where $v = (v_{11}, v_{12}, v_{13})$ controls the shape deformation. The last row of P is not $[0, 0, 1]$, which causes the shape deformation. We cannot use this projective transformation on primitive shapes as in Pictionary games, where one player builds an object part by part, and the other has to recognize the parts. If the shapes get deformed, they lose their recognizability, which motivates us to restrict transformations to an affine transformation.

9. Primitive based canvas completion

Problem Statement: Primitive-based canvas completion performs the same synthesis through optimization in a more constrained environment. A primitive shape is a special form of Bézier curves $\{S_i\}_{i=1}^n = \{s_i, w_i, \gamma_i\}_{i=1}^n$, where $S_i = \{s_i\}_{j=1}^p = \{x_i, y_i\}_{j=1}^p; \forall p \in \{2, 3, 4\}$ and they

can only go through the affine transformation (6 dof: rotation, translation and anisotropic-scaling) during optimization with $\mathcal{R}_d$. This effect of transformation constraints has been provided in Fig. 6, Tab. 4 and Fig. 15, Fig. 16.

As we proposed primitive-based scene completion as our downstream task, we decided to study the evolution of the primitive shapes without per-box mask latent base canvas initialization (i.e., without semantic guidance). This experiment aims to understand the true potential of primitive shapes toward canvas completion in a restricted environment with minimal guidance or no guidance at all. We have decided to initialize the canvas with primitive shapes in a random manner, as semantic guidance provides a strong initialization and shape guidance during iterative evolution, which significantly forces the shape evolution to maintain the semantic pattern. The results we have obtained via this experiment have been depicted in Fig. 15.

The semi-circle obtained the most aesthetically pleasing SVG (mean Aesthetic score: 5.0241), maintaining the text prompt due to open-ended and fewer control points (only 2), consolidating the claim we have obtained in 4 of the main paper. Similarly, the square shape provides the worst result due to its closed nature and higher number of control points (i.e., 4), constraining its transformation flexibility. However, we have an important observation in circle evolution, which produces a messier SVG compared to the one with semantic guidance. As the circle has a closed-form shape with only 2 control points, it has significantly less flexibility than a semicircle during evolution. So, during optimization without semantic guidance, it is unable to optimize the unwanted shapes initialized randomly.

However, from the Tab. 4, and Fig. 15 of this supplementary, it can be concluded that circle, semi-circle, and lines are the best possible shapes to complete a canvas, and square and triangle are the worst ones. However, we are keen to know if the shapes with a higher number of control points (square, triangle, and so on) can help the other shapes (circle, semi-circle, and line) in an auxiliary manner to generate more aesthetically promising SVGs. The outcomes of this study have been depicted in Fig. 16.

As shown in Fig. 16, the combination of circle+triangle improves the aesthetic score (4.0832 to 4.9163) from the SVG generated with only circles. Not only that, we can observe a certain failure for the rest of the combinations as the synthesized SVG contains a lot of unnecessary strokes that are not properly optimized during evolution. As circles and triangles have more inherent geometric compatibility compared to other combinations, the circular shape can easily accommodate the angular structure of a triangle without any overlapping or awkward intersections. Other combinations, like line and square, might not fit together as neatly due to differences in shape and curvature. Also, the circle-triangle combination creates a perception of symmetry or balance that the other combinations lack. Symmetry is often considered aesthetically pleasing, and the circular shape paired with the triangular shape creates a sense of balance that is missing in the other combinations.

9.1. Details of user study

We have conducted the perceptual study in order to evaluate the credibility and widespread adoption of the synthesized SVGs. We chose 60 SVGs synthesized by CLIPDraw (b/w and color) [1], CLIPDrawX [35], DiffSketcher [4], VectorFusion [3], SVGDreamcer [26], and CraftSVG (abstract, sketch, primitives, and CLIPArt), each style contains 10 examples. So we provide the users with 10 sets, each set containing 13 images, followed by the aforementioned style, and ask them to rate the SVGs on the scale of 1 to 5 (1 Low; 5 high) based on aesthetics, simplicity, and recognizability.

For the user, color is proportional to the aesthetically promising; that's why CraftSVG (CLIPArt) and SVGDreamer hold the 1st and 2nd positions, respectively. Similarly, they consider simple sketches as freehand drawings, which makes CraftSVG (abstract) the winner of this category, and CraftSVG (sketch) stands second. However, the 3rd question helps us to get the confusion score (i.e., 100 - % of people identify the image as AI generated). CraftSVG (abstract) is a clear winner, and CraftSVG (sketch) stands second in this category as the light-shedding gives an artistic touch to the synthesized SVGs. All the users agreed that the SVGs synthesized by CraftSVG properly followed the complex text prompt with enumeration and spatial relationship, which consolidated our claim in this paper.

10. Importance of Alignment Loss

In this section, we have performed the ablation study of the CraftSVG with and without using the perceptual alignment loss defined in the Eq. (6). It can be done by setting the $\lambda_{align} = 0$ while optimizing the CraftSVG with $\mathcal{L}_{synth}$ as defined in Eq. (8). The result of this ablation study has been reported in Fig. 18 and Tab. 5, respectively.

From Fig. 18, it can be observed that with perceptual alignment loss, the background and foreground strokes often collide with the foreground strokes, leading to noisy abstract VectorArts that do not even follow the text prompt. On the other hand, with perceptual alignment loss, CraftSVG always preserves the foreground object structure while optimizing the background strokes during alignment. Also, Tab. 5 depicts a significant performance improvement of CraftSVG while using the perceptual alignment loss. Also, it can help to optimize the SVG parameters along with the MLP parameters jointly to abstractify the VectorArt by maintaining enumeration, spatial arrangement, and relationship provided by a text prompt for complex scenes.

Table 4. Evaluation of canvas completions with primitive shapes.

Shapes/Metric	CS ↑ [48]	FID ↓ [49]	PSNR ↑ [50]	CLIP-T ↑ [2]	BLIP ↑ [51]	Aes. ↑ [36]	HPS ↑ [37]	Conf. ↑ [4]
Circle	0.4817	76.38	10.24	0.2974	0.4265	4.7531	0.2231	0.39
Line	0.4566	82.37	7.62	0.2369	0.3664	4.1522	0.1976	0.32
Semi-Circle	**0.5194**	**61.34**	**12.22**	**0.3145**	**0.4578**	**5.1362**	**0.2413**	**0.47**
Triangle	0.2237	111.54	2.11	0.1433	0.2237	3.0567	0.1032	0.24
Square	0.2013	127.83	1.07	0.1123	0.1952	2.3645	0.0914	0.11
L-shape	0.3176	94.56	5.67	0.1958	0.2987	3.7134	0.1532	0.29
U-shape	0.3124	96.82	4.12	0.1663	0.2637	3.228	0.1338	0.24

Table 5. Ablation of the Alignment Loss

Methods/Metrics	CS	FID	PSNR	CLIP-T	BLIP-T	Aes	HPS
w/o alignment loss	0.2181	122.62	4.33	0.1484	0.2575	3.2863	0.1877
With alignment loss	0.6176	51.42	15.98	0.4563	0.5223	5.9873	0.3167

11. Attention-based initialization comparison

All the diffusion-based text-to-SVG synthesis techniques first synthesize a guided image from the text prompt and then optimize the strokes/blobs in the canvas by maximizing the similarity with the guided image. It has been interesting to study how attention-based initialization makes a difference in the final SVG and helps in faster convergence (see Fig. 19, Fig. 20, Fig. 21). Diffsketcher [4] utilizes the diffusion U-Net map on the synthesized image to identify the keypoints for stroke initialization. Although it helps in faster convergence, it is unable to follow the enumeration and spatial relationship mentioned in the input text prompt. Similarly, SVGDreamer creates separate semantic maps for foreground and background objects using the text encoder's cross-attention map on the synthesized guided image, optimizing them independently with attention mask-based loss. This reduces interaction between elements, making the drawing appear AI-generated. It has to be noted that both methods first create the synthesized guided image and then try to get the attention map, which is unable to control the object counting and their position in the canvas. On the contrary, CraftSVG first generates a layout from the input text prompt via LLM in order to follow the enumeration and spatial relationship mentioned in the prompt. Then, we use per-box mask latent guidance, processing a single foreground object box through the diffusion U-Net to extract coherent foreground and background masks. By iteratively merging these masks, CraftSVG maintains depth information, allowing opacity control, where strokes of closer objects have higher opacity. Also, we initialize strokes only in the foreground region and then optimize the length of the strokes along with their width and color by maximizing the similarity with the synthesized guided image obtained through layout guidance.

12. Comparison with T2I diffusion models

With the recent advent of text-to-image diffusion models [17, 27], [28] we are keen to know about their potential to replicate freehand drawing style. In order to perform this evaluation we compare the SVGs synthesized by CraftSVG with the image generated by the diffusion models. In order to encourage the results to be abstract and follow the free-hand sketch style, we append a suffix to the text prompt: "A free-hand sketch of xxx" as mentioned in [4]. The obtained qualitative as well as quantitative results are reported in Fig. 22 and Tab. 6 respectively. As depicted in Fig. 22, LDM [5] and Richtext2image [28] are unable to follow the proper enumeration and spatial relationship mentioned in the text prompt. In Fig. 22(c), LDM generated a real image, and in Fig. 22(b), Richtext2image draws a boat in the sky, which is beyond specification. Although LLM-grounded diffusion [27] follows the text prompt, it is unable to make the freehand sketch effect showing the importance as well as the superiority of our proposed CraftSVG. It not only follows the enumeration and spatial relationship but also creates the specific style mentioned by the user.

Similarly, we have observed an interesting fact in Tab. 6. The aesthetics of Richtext2image is slightly higher than the proposed CraftSVG as it generates high-quality pixel graphics (1024×1024), whereas LDM, LLM-grounded diffusion, as well as CraftSVG work on the canvas size of 512×512. Although Richtext2image generates high-quality images, its background exhibits an old paper texture, which looks like an old grayscale photograph rather than sketches. This comes from the bias of the training dataset used by Richtext2image. Conversely, the sketches generated by our CraftSVG bear a closer resemblance to free-hand sketching styles and display varying degrees of abstraction.

13. Ablation of the image encoders

As we perform similarity maximization between the encoded features of the rasterized target image $\mathcal{I}_r$ and the differential renderer $\mathcal{R}_d(\theta)$, we should stress about choosing the best feature extractor. However, there are no such metrics to compare the feature extraction performance between different backbones. In order to compare the different image encoders $\mathcal{I}$, we have tried to extract their activation map/ attention map of the last layers for Convolution/transformer-based architectures. The obtained results have been reported in Fig. 23. As we can observe in Fig. 23, the Grad-CAM results of the last layer of VGG19 don't capture all the required features to complete the scene, leading to the synthesized incomplete sketches. With ResNet-50, we can improve the performance, however, it only focuses on the main objects, not on the background scenarios. Moreover, with the attention used in the

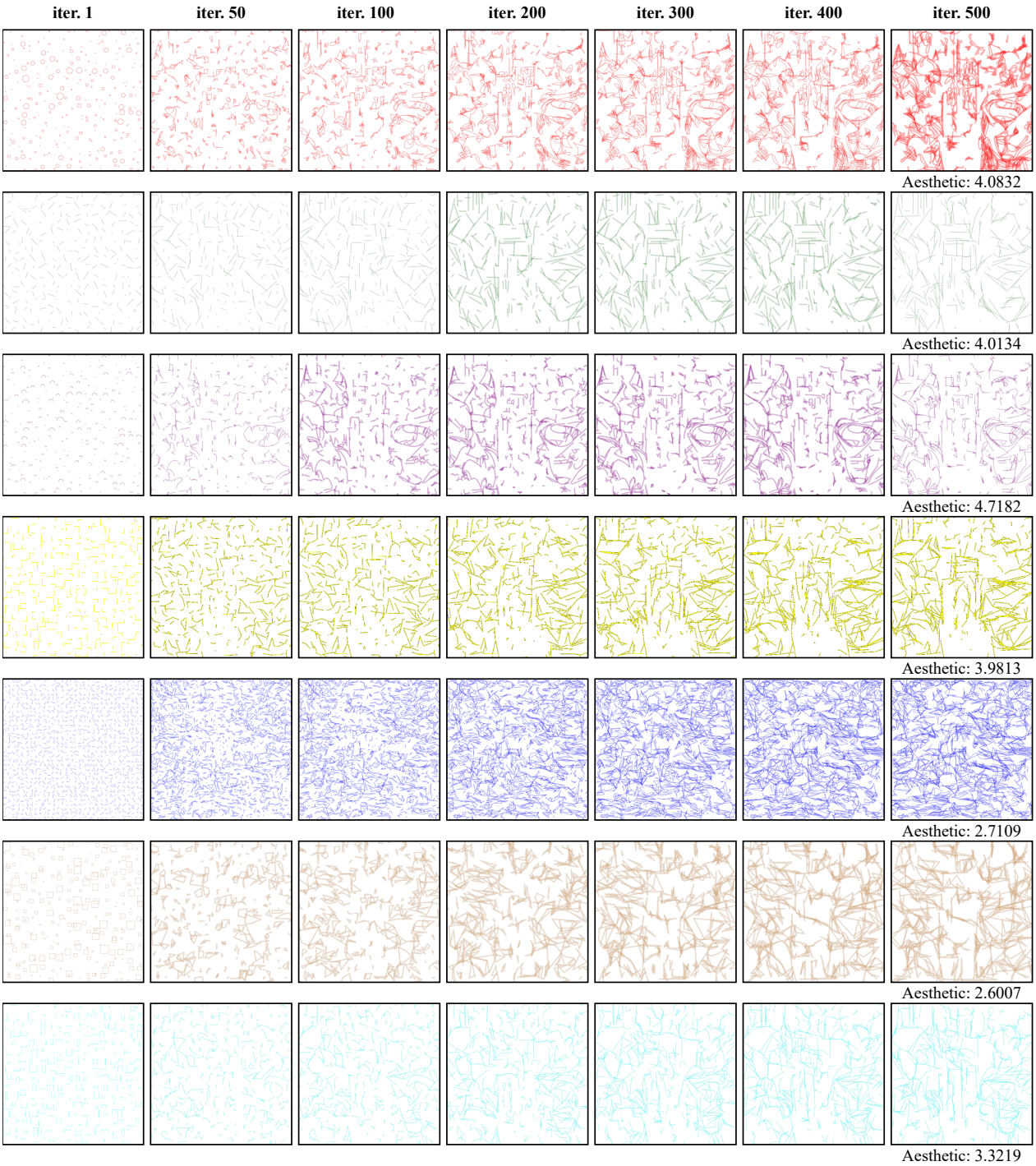


Fig. 15. **Evolution of the primitive shapes:** circle, line, semi-circle, L-shape, triangle, square, and U-shape with random initialization for the prompt "Batman and Superman, with smoke and explosions in the background". Over the iteration semi-circle generates the best quality SVGs compared to the rest.

Table 6. Quantitative evaluation on conventional text-to-image synthesis techniques.

Method / Metric	CS ↑ [48]	FID ↓ [49]	PSNR ↑ [50]	CLIP-T ↑ [2]	BLIP ↑ [51]	Aes. ↑ [36]	HPS ↑ [37]
LDM [23]	0.4906	99.18	12.24	0.3512	0.4221	4.9317	0.2332
Richtext2image [28]	0.4130	87.04	14.34	0.3772	0.4017	**7.6468**	0.2661
LLM-grounded diffusion [27]	0.5052	69.18	15.08	0.3815	0.4327	5.9821	0.2858
CraftSVG (Sketch)	0.6342	48.42	16.07	0.4563	0.5223	6.7832	0.3167
CraftSVG (CLIPArt)	**0.7091**	**39.87**	**17.15**	**0.5013**	**0.5783**	7.0779	**0.3523**

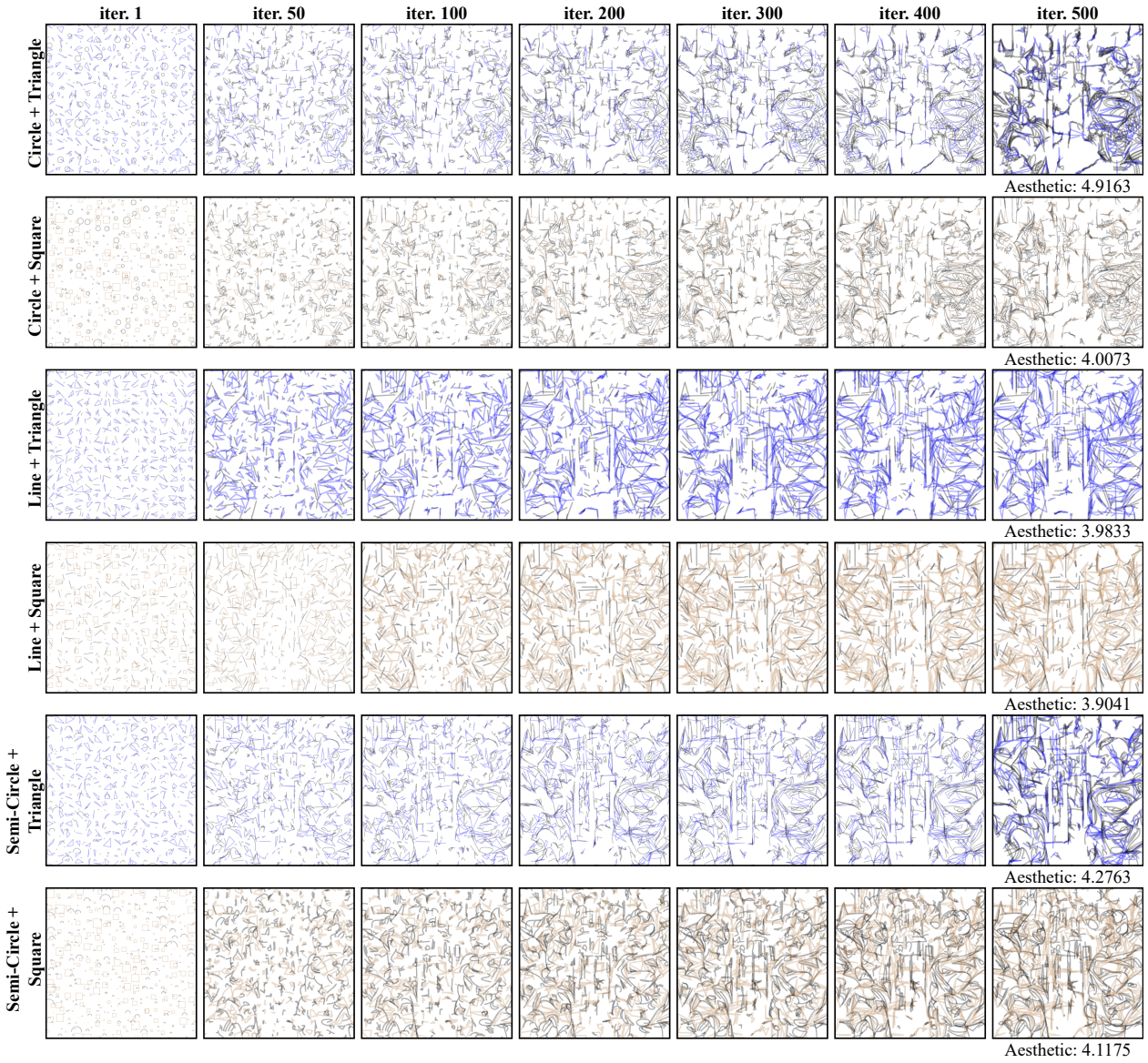


Fig. 16. **Mixing of primitive shapes:** We have used circles, semi-circles, and lines as our base shape with triangles and squares as an auxiliary one. Although it improves the aesthetic score for circle+triangle, it provides messier SVGs for the rest.

ConVMixer model [59]s, we can capture the global perspective but miss the local minute details. So we have decided to use the CLIP ViT B-32, which divides the input image into patches to get the global perspective and uses transformer layers to capture the local one, leading to use this as our desired image encoder in CraftSVG.

14. Effect of bounding box scaling

Vector graphics synthesis operates solely within the bounding box, often producing results that are overly constrained and lack contextual coherence beyond the specified boundaries. This approach may lead to incomplete or fragmented images that fail to capture the broader scene or object context, limiting the model's ability to generate creative and visually appealing outputs. In order to address this issue, we have introduced bounding box scaling techniques, which enable us to use the full canvas for generating objects instead of the specified boxes. Fig. 24 demonstrates the SVGs synthesized with and without bounding box scaling, keeping the other attributes (text prompt, layout, no. of strokes, stroke width, opacity, and so on) unaltered.

In Fig. 24, scaling enables the incorporation of surrounding elements and spatial relationships, enriching the generated images with more coherent and contextually relevant details. This broader perspective enhances the model's

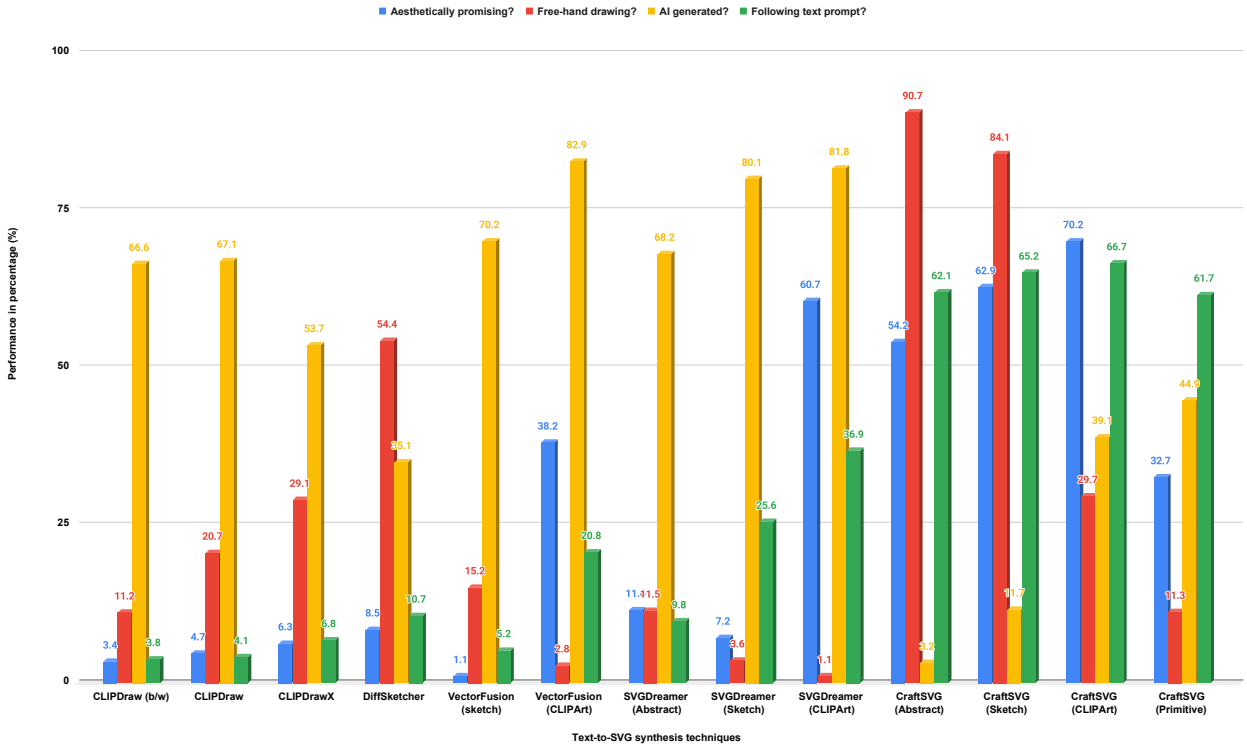


Fig. 17. **User study:** CraftSVG gets the maximum vote for being aesthetically promising, whereas, VectorFusion is identified as AI generated due to excessive smoothing and color over-saturation.

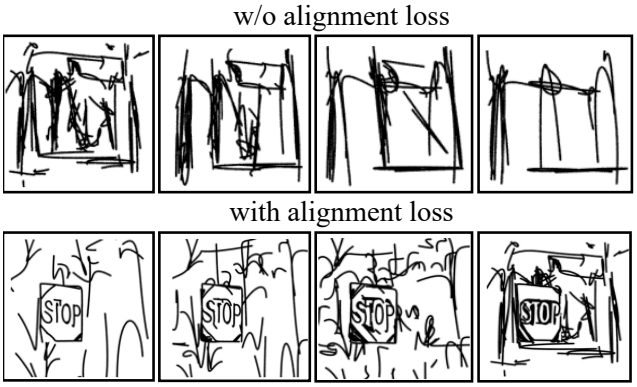


Fig. 18. The ablation study of the alignment loss with the text prompt "The stop sign behind the fence"

capacity to generate visually convincing and semantically meaningful images that better reflect objects on the canvas. However, some results can be penalized as they cut out the face in the 2nd and 6th columns of Fig. 24 as the amount of the scaling is controlled by the model implicitly. Still, scaling bounding boxes offers a more creative approach to vector graphics generation, facilitating the creation of more immersive and contextually rich visual outputs.

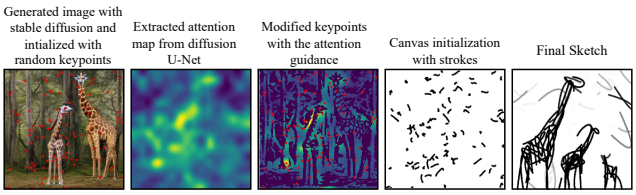


Fig. 19. Attention-based initialization used in the DiffSketcher [4]. For the prompt "Two giraffes and three elephants", it directly initializes the keypoints from the attention map diffusion U-Net.

15. Applications

The main application of SVG generation lies in creative logo design and creative poster design. We have also added another domain of application, namely, simple architecture design, as shown in Fig. 25. It has been observed that the proposed CraftSVG (third row of Fig. 25) can produce an effective architectural design mentioned in a text prompt.

Not only that, most of the recent work mentioned the issues of text generation in the poster as well as creative logo design [23] [60]. In order to address this issue, SVG-Dreamer [26] utilized an additional text style generation module, GlyphControl [60], which maintains the coherency between the visual and textual content of the poster. On the

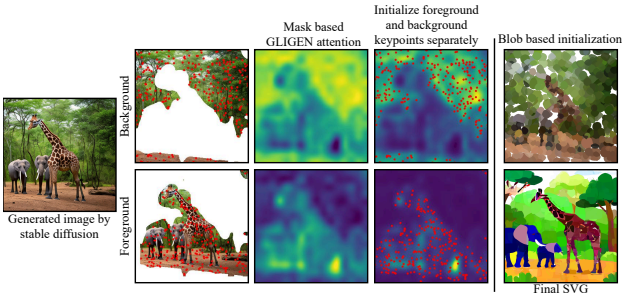


Fig. 20. Attention-based initialization used in the SVGDreamer [26]. For the prompt "Two giraffes and three elephants in a forest", it creates the foreground mask to separate the objects from the background and initialize blobs of those two regions separately.

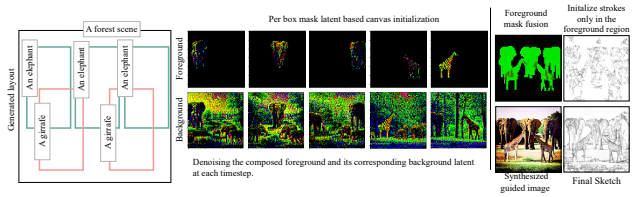


Fig. 21. Attention-based initialization used in CraftSVG. For the prompt "Two giraffes and three elephants in a forest", it follows a progressive approach via layout guidance and initializes strokes only in the foreground region, and optimizes stroke length along with its color and width to complete the canvas.

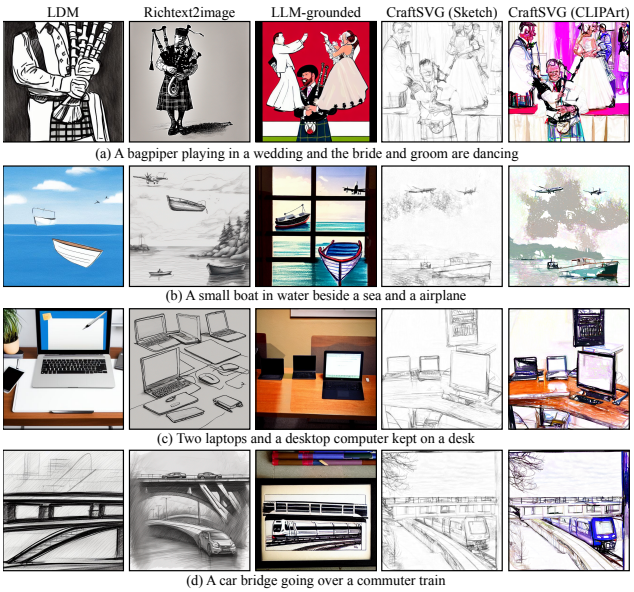


Fig. 22. Comparison with text-to-image synthesis techniques.

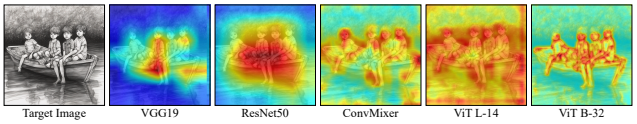


Fig. 23. Comparison of different types of image encoders based on their feature extraction performance.

other hand, CraftSVG can successfully generate visual as well as textual content without any additional help.

16. More qualitative examples

This section is dedicated to demonstrating the diverse power of the CraftSVG by generating diverse SVGs from the same as well as different text prompts. In order to enhance the readability, we have further categorized the qualitative examples in the following subsections. It has to be noted that, all the examples have been generated in the following three styles: sketch, CLIPArt, and primitive shape. As it is already proven that, circles, semi-circles, and lines generate more aesthetically promising SVGs, we use the combinations of these three types of shapes to complete the canvas. We make some exceptions to this rule in order to produce variability in the demonstration.

Imaginary object synthesis: This section deals with imaginary objects (*e.g.*, dragon, unicorn, yeti, baby Yoda, and so on) synthesized by CraftSVG. The qualitative results have been reported in Fig. 26. It is interesting to see the creativity of the CraftSVG to make unicorns in purple color, which is beyond imagination as in most of the existing examples, a unicorn is usually generated in white color.

Spatial Arrangement: Here, we try to synthesize SVGs maintaining the spatial relationship (right, left, above, and so on). Fig. 27 demonstrates the performance of the CraftSVG with complex spatial arrangement prompts. This shows that CraftSVG has the potential to synthesize the SVGs following the arrangement described in the prompt.

Details of single object: Object detailing is one of the crucial components of human drawing as it is one of the significant ways to demonstrate the artist's capability to reproduce reality in detail. In order to replicate the same in synthesizing SVGs, we fed CraftSVG some prompts containing only single objects with specified details. As presented in Fig. 28, CraftSVG demonstrates the ability to create detailed and precise objects, with the exception of facial features. Investigating the latter aspect further could serve as a compelling avenue for future research.

Miscellaneous Here we mixed all types of prompts with enumeration, spatial relationship, detailing, behavioral expression, and so on. From Fig. 29, it can be concluded that CraftSVG successfully tackles all the complexity provided in the text prompt.

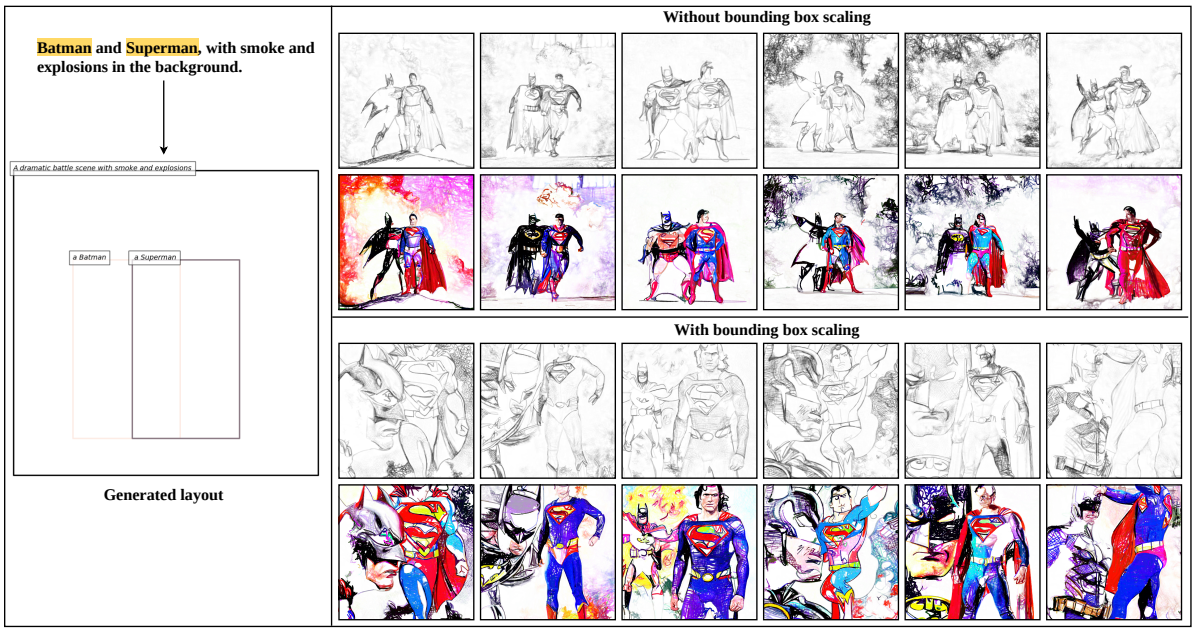


Fig. 24. **Effect of bounding box scaling:** it enables the CraftSVG to create more creative, visually appealing SVGs, maintaining the contextual information provided in the text prompt by relaxing the constrained area specified in a bounding box.

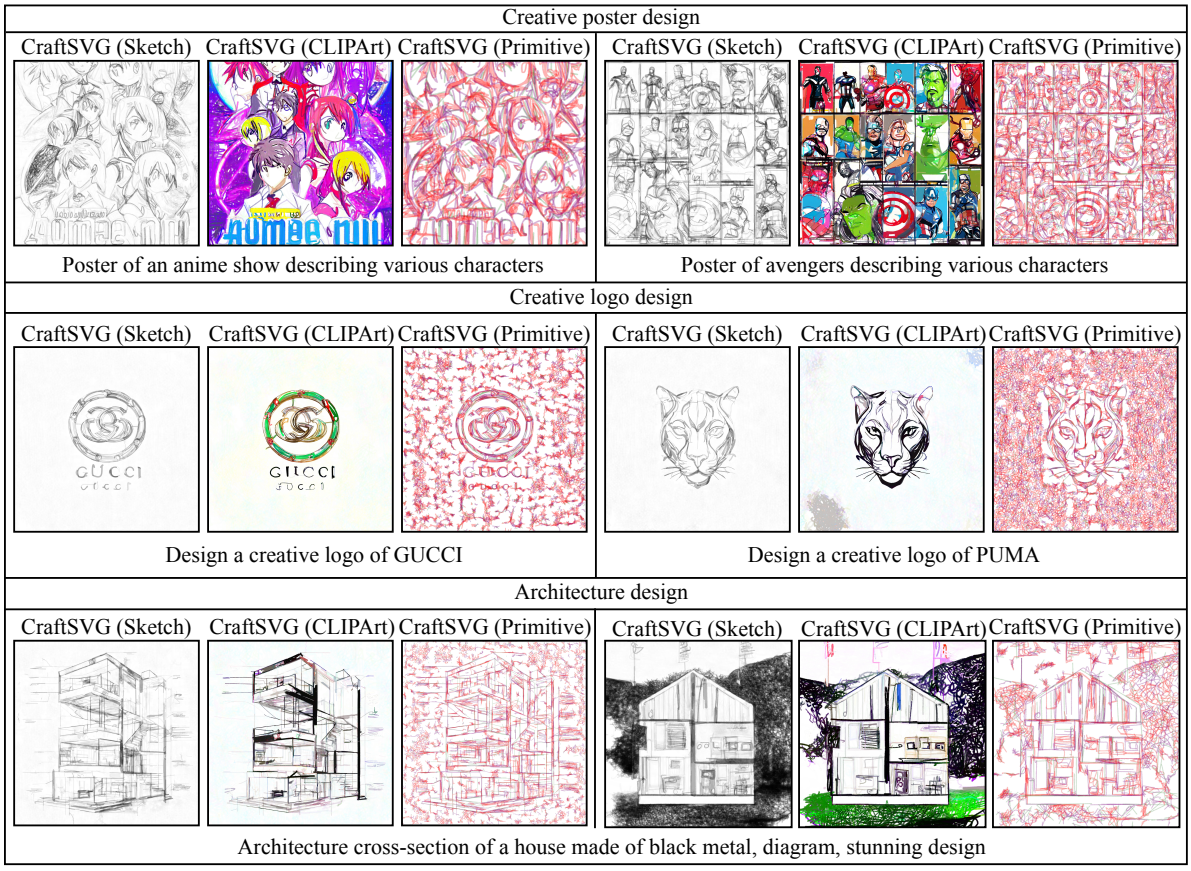
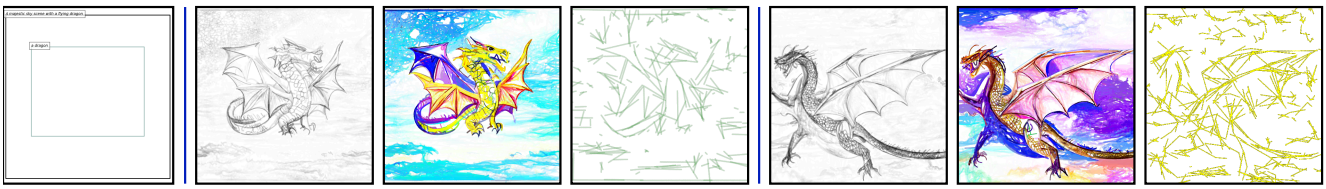
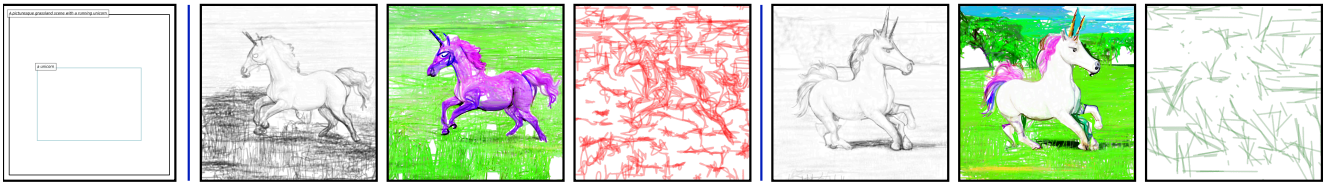


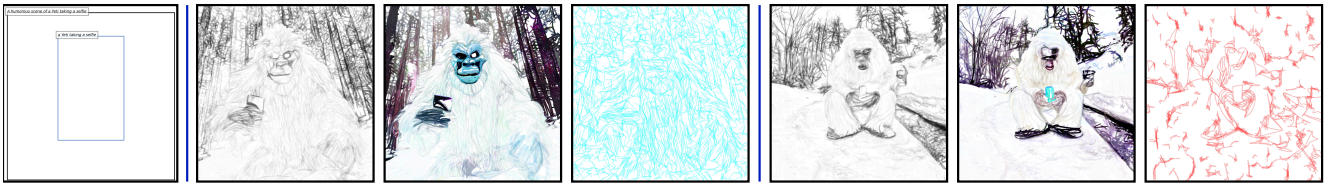
Fig. 25. **Applications of CraftSVG:** It can design simple architecture besides creative posters and logos without any additional help from the text style generation module.



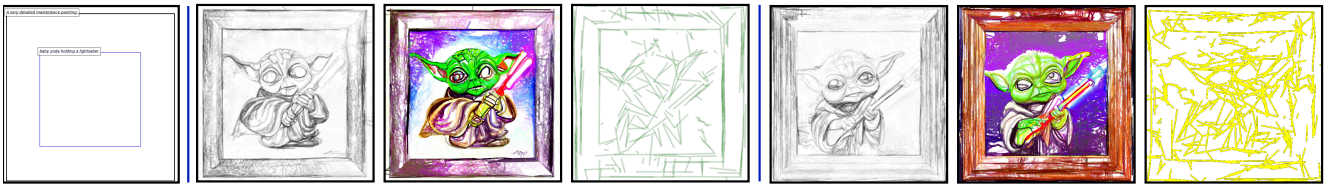
(a) A dragon flying in the sky, full body



(b) A unicorn is running on the grassland

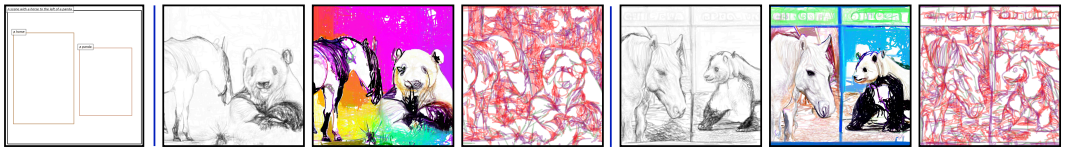


(c) A yeti is taking a selfie

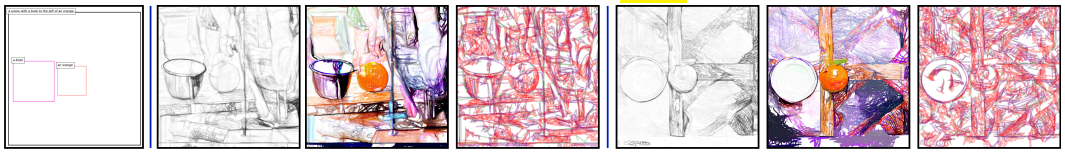


(d) Very detailed masterpiece painting of baby yoda holding a lightsaber

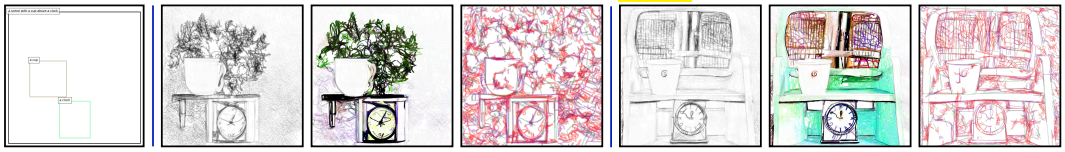
Fig. 26. Non-existance object synthesis with CraftSVG



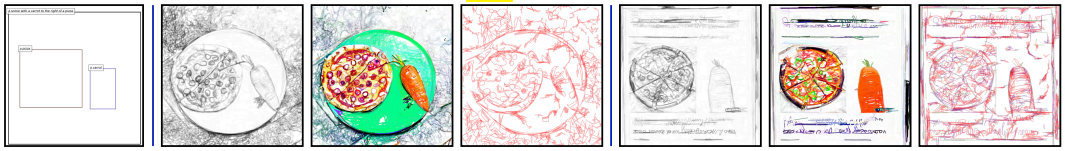
(a) A horse to the left of a panda



(b) a bowl to the left of an orange

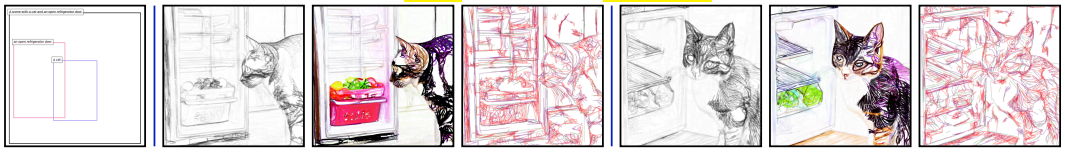


(c) A cup above a clock

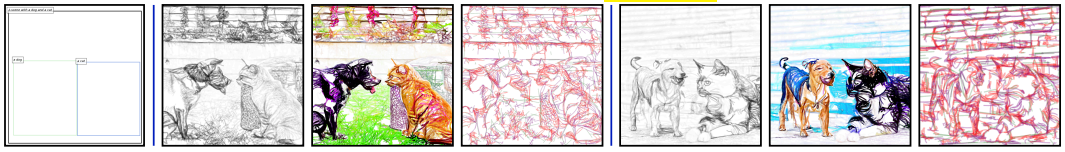


(d) A carrot to the right of a pizza

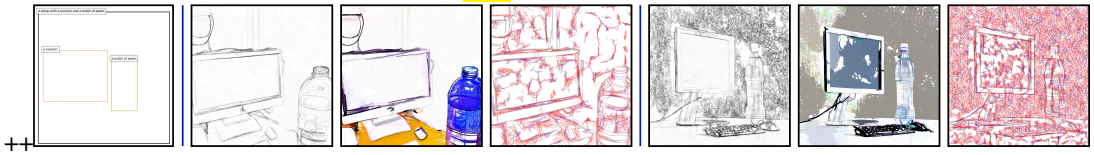
(e) A bowl to the right of a snowboard



(f) a cat standing next to an open refrigerator door

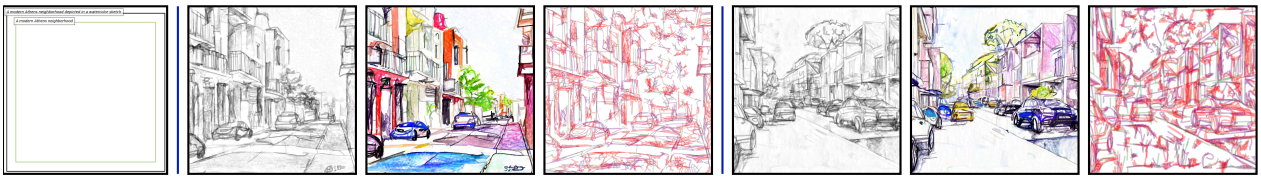


(g) A dog to the left of a cat

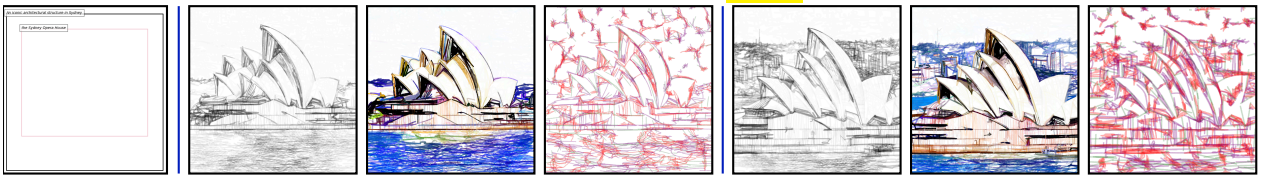


(h) A bottle of water is placed to the right of a monitor

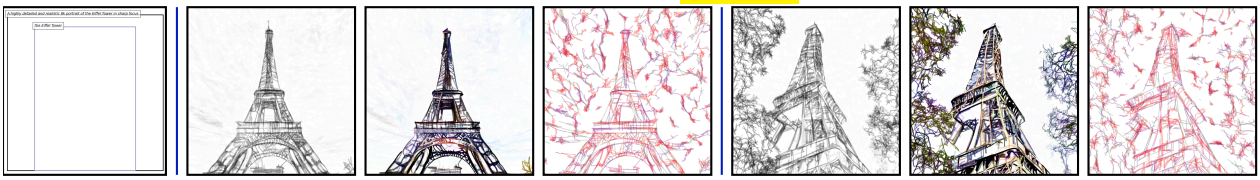
Fig. 27. **Spatial arrangement:** CraftSVG can easily determine the object position following the direction left, right, above, next and so on.



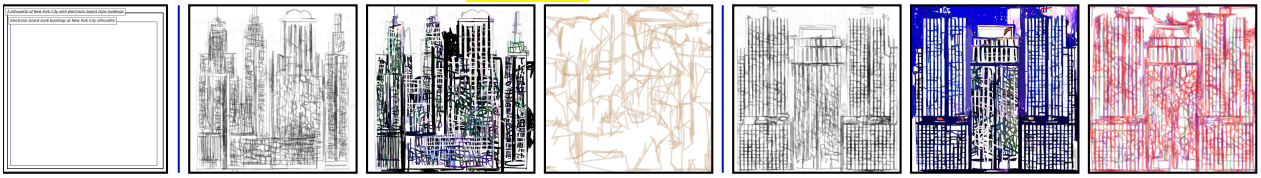
(a) A sketching with watercolors of a modern Athens neighborhood



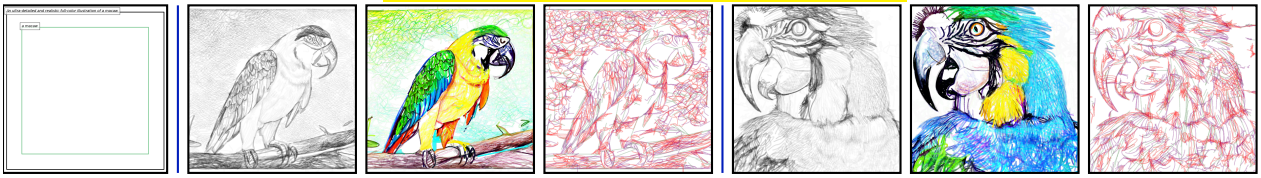
(b) Real photo of Sydney opera house



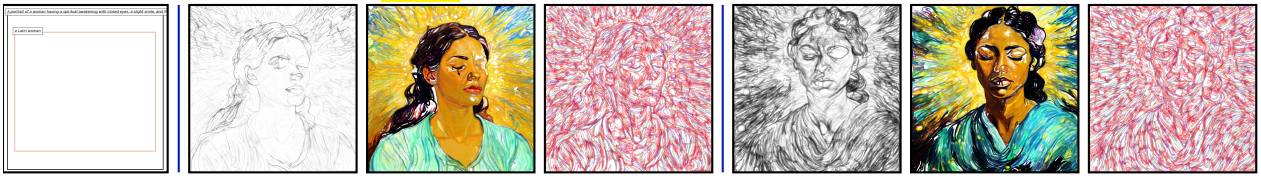
(c) A detailed portrait of Eiffel Tower, incredibly detailed and realistic, 8k, sharp focus



(d) Electronic board style buildings at New York city silhouette



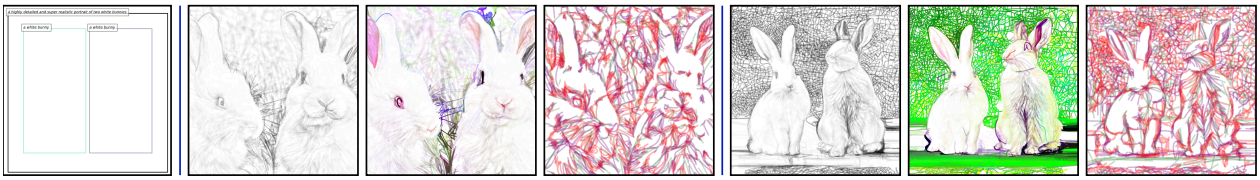
(e) Macaw full color, ultra-detailed, realistic, insanely beautiful



(f) Potrait of Latin woman having a spiritual awakening, eyes closed, slight smile, illuminating lights, oil painting, by Van Gogh

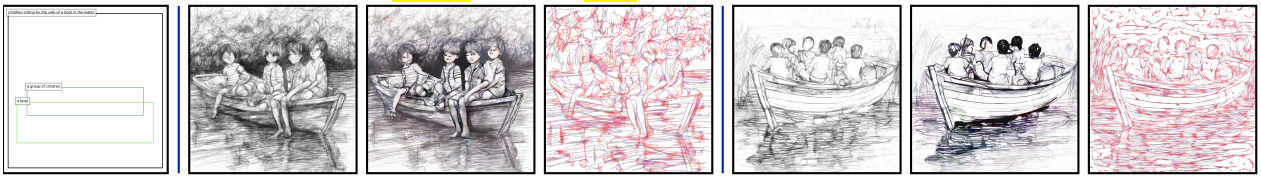
(g) A tiny analog clock

Fig. 28. **Object detailing:** CraftSVG produces detailing with user-specific style except for the human faces.

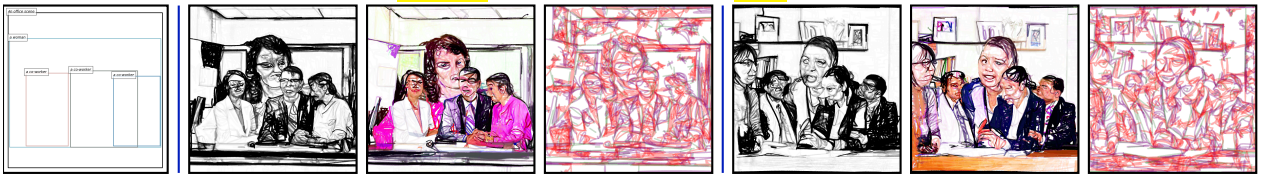


(a) Portrait of two white bunnies, super realistic, highly detailed

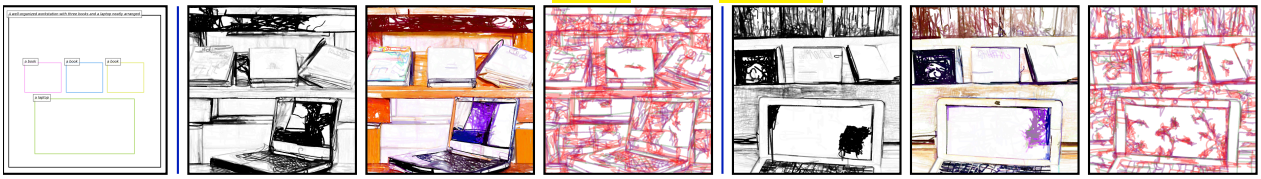
(b) A woman is riding her bike down the street in front of traffic



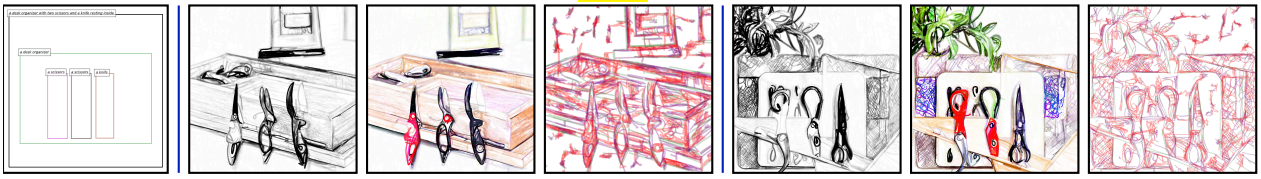
(c) Children are sitting on the side of the boat in the water



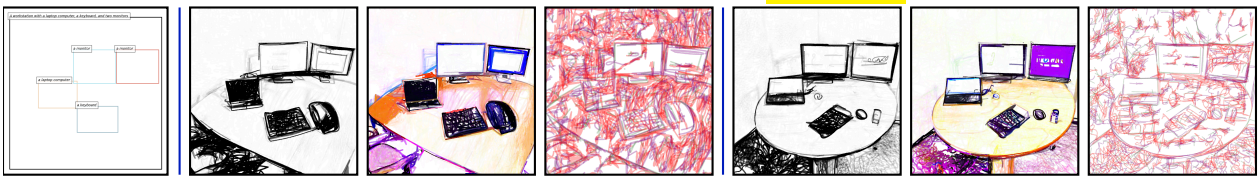
(d) An angry woman staring at coworkers



(e) Three books and a laptop kept neatly in a workstation



(f) A pair of scissors and a knife resting in a desk organizer



(g) A laptop computer a keyboard and two monitors

Fig. 29. **Miscellaneous:** with different types of descriptions CraftSVG successfully produces the desired SVGs specified in the text prompt.